

MQL5 Language REFERENCE

for the MetaTrader 5 client terminal

STUDY MQL5 and SOLVE any tasks:

- Create your own technical analysis indicators of any complexity
- Use autotrading - automate trading systems to work on various financial markets
- Develop your own analytical tools based on mathematical achievements and traditional methods
- Write information trading systems for solving a wide range of tasks (trading, monitoring, alerting, etc.)

Content

MQL5 Reference

60

1 Language Basics.....	62
Syntax	63
Comments	64
Identifiers.....	65
Reserved Words.....	66
Data Types	68
Integer Types.....	69
Char, Short, Int and Long Types	70
Character Constants.....	74
Datetime Type.....	77
Color Type	78
Bool Type	79
Enumerations	80
Real Types (double, float).....	82
String Type.....	87
Structures, Classes and Interfaces.....	88
Dynamic Array Object.....	101
Typecasting	102
Void Type and NULL Constant.....	109
User-defined Types.....	110
Object Pointers.....	120
References: Modifier & and Keyword this.....	123
Operations and Expressions	125
Expressions.....	126
Arithmetic Operations.....	127
Assignment Operations.....	128
Operations of Relation.....	129
Boolean Operations	130
Bitwise Operations.....	132
Other Operations	135
Precedence Rules.....	139
Operators	141
Compound Operator	143
Expression Operator.....	144
Return Operator.....	145
Conditional Operator if-else.....	146
Ternary Operator ?:	147
Switch Operator.....	149
Loop Operator while	151
Loop Operator for.....	152
Loop Operator do while	153
Break Operator	154
Continue Operator	155
Object Create Operator new.....	156
Object Delete Operator delete.....	157
Functions	158
Function Call.....	160
Passing Parameters	161
Function Overloading.....	164
Operation Overloading.....	167
Description of External Functions.....	180
Exporting Functions	182

Event Handling Functions	183
Variables	194
Local Variables.....	196
Formal Parameters	198
Static Variables.....	200
Global Variables	202
Input Variables.....	203
Extern Variables.....	207
Initialization of Variables	208
Visibility Scope and Lifetime of Variables	210
Creating and Deleting Objects.....	212
Preprocessor	215
Macro substitution (#define).....	216
Program Properties (#property).....	219
Including Files (#include).....	224
Importing Functions (#import).....	225
Conditional Compilation (#ifdef, #ifndef, #else, #endif)	227
Object-Oriented Programming	229
Encapsulation and Extensibility of Types.....	231
Inheritance.....	234
Polymorphism.....	239
Overload	243
Virtual Functions	244
Static Members of a Class.....	248
Function templates.....	252
Class templates	256
Abstract Classes.....	261
2 Standard Constants, Enumerations and Structures	263
Chart Constants	264
Types of Chart Events.....	265
Chart Timeframes	271
Chart Properties	273
Positioning Constants	280
Chart Representation	281
Examples of Working with the Chart.....	283
Objects Constants	341
Object Types.....	342
OBJ_VLINE	344
OBJ_HLINE	349
OBJ_TREND	354
OBJ_TRENDBYANGLE	361
OBJ_CYCLES.....	367
OBJ_ARROWED_LINE.....	373
OBJ_CHANNEL.....	379
OBJ_STDDEVCHANNEL.....	386
OBJ_REGRESSION.....	393
OBJ_PITCHFORK.....	399
OBJ_GANNLIN.....	407
OBJ_GANNFAN.....	414
OBJ_GANNGRID.....	421
OBJ_FIBO	428
OBJ_FIBOTIMES	435
OBJ_FIBOFAN.....	442
OBJ_FIBOARC.....	449
OBJ_FIBOCHANNEL.....	456
OBJ_EXPANSION.....	464
OBJ_ELLIOTWAVE5.....	472
OBJ_ELLIOTWAVE3.....	480

OBJ_RECTANGLE	487
OBJ_TRIANGLE.....	493
OBJ_ELLIPSE.....	500
OBJ_ARROW_THUMB_UP.....	506
OBJ_ARROW_THUMB_DOWN.....	512
OBJ_ARROW_UP.....	518
OBJ_ARROW_DOWN.....	524
OBJ_ARROW_STOP.....	530
OBJ_ARROW_CHECK.....	536
OBJ_ARROW_LEFT_PRICE	542
OBJ_ARROW_RIGHT_PRICE.....	547
OBJ_ARROW_BUY	552
OBJ_ARROW_SELL	557
OBJ_ARROW.....	562
OBJ_TEXT	568
OBJ_LABEL	574
OBJ_BUTTON.....	582
OBJ_CHART	589
OBJ_BITMAP.....	596
OBJ_BITMAP_LABEL.....	603
OBJ_EDIT	610
OBJ_EVENT	617
OBJ_RECTANGLE_LABEL	622
Object Properties.....	628
Methods of Object Binding.....	636
Chart Corner.....	641
Visibility of Objects.....	644
Levels of Elliott Wave.....	647
Gann Objects.....	648
Web Colors.....	650
Wingdings.....	652
Indicator Constants	653
Price Constants.....	654
Smoothing Methods.....	657
Indicators Lines	658
Drawing Styles	660
Custom Indicator Properties.....	664
Indicator Types.....	667
Data Type Identifiers.....	669
Environment State	670
Client Terminal Properties.....	671
Running MQL5 Program Properties.....	676
Symbol Properties	679
Account Properties	692
Testing Statistics.....	696
Trade Constants	700
History Database Properties.....	701
Order Properties	702
Position Properties.....	706
Deal Properties	709
Trade Operation Types.....	712
Trade Transaction Types.....	724
Trade Orders in DOM	727
Signal Properties.....	728
Named Constants	730
Predefined Macro Substitutions	731
Mathematical Constants	733
Numerical Type Constants	735

Uninitialization Reason Codes.....	738
Checking Object Pointer.....	740
Other Constants.....	741
Data Structures	745
Date Type Structure.....	746
Indicator Parameter Structure.....	747
History Data Structure.....	748
Order Book Structure.....	749
Trade Request Structure.....	750
Request Check Result Structure.....	764
Trade Request Result Structure.....	765
Trade Transaction Structure.....	769
Price Data Structure.....	777
Codes of Errors and Warnings	778
Trade Server Return Codes.....	779
Compiler Warnings.....	782
Compilation Errors.....	786
Runtime Errors.....	798
Input/Output Constants	808
File Opening Flags.....	809
File Properties.....	811
In-File Position.....	812
Use of a Codepage.....	813
MessageBox.....	814
3 MQL5 programs.....	816
Program Running	817
Trade Permission	823
Client Terminal Events	827
Resources	830
Call of Imported Functions	842
Runtime Errors	844
Testing Trading Strategies	845
4 Predefined Variables	871
_Digits	872
_Point	873
_LastError	874
_Period	875
_RandomSeed	876
_StopFlag	877
_Symbol	878
_UninitReason	879
5 Common Functions.....	880
Alert	882
CheckPointer	883
Comment	885
CryptEncode	886
CryptDecode	888
DebugBreak	889
ExpertRemove	890
GetPointer	892
GetTickCount	896
GetMicrosecondCount	897
MessageBox	899
PeriodSeconds	900
PlaySound	901
Print	902
PrintFormat	904

ResetLastError	910
ResourceCreate	911
ResourceFree	913
ResourceReadImage	914
ResourceSave	915
SetUserError	916
SendFTP	917
SendNotification	918
SendMail	919
Sleep	920
TerminalClose	921
TesterStatistics	923
TesterWithdrawal	924
TranslateKey	925
WebRequest	926
ZeroMemory	932
6 Array Functions	933
ArrayBsearch	934
ArrayCopy	938
ArrayCompare	943
ArrayFree	944
ArrayGetAsSeries	953
ArrayInitialize	956
ArrayFill	958
ArrayIsDynamic	960
ArrayIsSeries	962
ArrayMaximum	964
ArrayMinimum	975
ArrayPrint	986
ArrayRange	989
ArrayResize	990
ArraySetAsSeries	993
ArraySize	996
ArraySort	998
7 Conversion Functions.....	1003
CharToString	1005
CharArrayToString	1006
ColorToARGB	1007
ColorToString	1009
DoubleToString	1010
EnumToString	1011
IntegerToString	1013
ShortToString	1014
ShortArrayToString	1015
TimeToString	1016
NormalizeDouble	1017
StringToCharArray	1019
StringToColor	1020
StringToDouble	1021
StringToInteger	1022
StringToShortArray	1023
StringToTime	1024
StringFormat	1025
8 Math Functions.....	1029
MathAbs	1031
MathArccos	1032
MathArcsin	1033

MathArcTan	1034
MathCeil	1035
MathCos	1036
MathExp	1037
MathFloor	1038
MathLog	1039
MathLog10	1040
MathMax	1041
MathMin	1042
MathMod	1043
MathPow	1044
MathRand	1045
MathRound	1046
MathSin	1047
MathSqrt	1048
MathSrand	1049
MathTan	1052
MathIsValidNumber	1053
MathExpM1	1054
MathLog1p	1055
MathArccosh	1056
MathArcsinh	1057
MathArcTanh	1058
MathCosh	1059
MathSinh	1060
MathTanh	1061
9 String Functions	1062
StringAdd	1063
StringBufferLen	1065
StringCompare	1066
StringConcatenate	1068
StringFill	1069
StringFind	1070
StringGetCharacter	1071
StringInit	1072
StringLen	1073
StringReplace	1074
StringSetCharacter	1075
StringSplit	1077
StringSubstr	1079
StringToLower	1080
StringToUpper	1081
StringTrimLeft	1082
StringTrimRight	1083
10 Date and Time	1084
TimeCurrent	1085
TimeTradeServer	1086
TimeLocal	1087
TimeGMT	1088
TimeDaylightSavings	1089
TimeGMTOffset	1090
TimeToStruct	1091
StructToTime	1092
11 Account Information	1093
AccountInfoDouble	1094
AccountInfoInteger	1095
AccountInfoString	1097

12	Checkup	1098
	GetLastError	1099
	IsStopped	1100
	UninitializeReason	1101
	TerminalInfoInteger	1102
	TerminalInfoDouble	1103
	TerminalInfoString	1104
	MQLInfoInteger	1105
	MQLInfoString	1106
	Symbol	1107
	Period	1108
	Digits	1109
	Point	1110
13	Market Info	1111
	SymbolsTotal	1112
	SymbolName	1113
	SymbolSelect	1114
	SymbolsSynchronized	1115
	SymbolInfoDouble	1116
	SymbolInfoInteger	1118
	SymbolInfoString	1120
	SymbolInfoMarginRate	1121
	SymbolInfoTick	1122
	SymbolInfoSessionQuote	1123
	SymbolInfoSessionTrade	1124
	MarketBookAdd	1125
	MarketBookRelease	1126
	MarketBookGet	1127
14	Timeseries and Indicators Access.....	1128
	Indexing Direction in Arrays, Buffers and Timeseries	1132
	Organizing Data Access	1135
	SeriesInfoInteger	1144
	Bars	1146
	BarsCalculated	1149
	IndicatorCreate	1151
	IndicatorParameters	1153
	IndicatorRelease	1155
	CopyBuffer	1157
	CopyRates	1162
	CopyTime	1166
	CopyOpen	1169
	CopyHigh	1172
	CopyLow	1176
	CopyClose	1179
	CopyTickVolume	1182
	CopyRealVolume	1186
	CopySpread	1189
	CopyTicks	1193
	CopyTicksRange	1199
15	Chart Operations.....	1201
	ChartApplyTemplate	1204
	ChartSaveTemplate	1207
	ChartWindowFind	1212
	ChartTimePriceToXY	1214
	ChartXYToTimePrice	1215
	ChartOpen	1217
	ChartFirst	1218

ChartNext	1219
ChartClose	1220
ChartSymbol	1221
ChartPeriod	1222
ChartRedraw	1223
ChartSetDouble	1224
ChartSetInteger	1225
ChartSetString	1226
ChartGetDouble	1227
ChartGetInteger	1229
ChartGetString	1231
ChartNavigate	1233
ChartID	1236
ChartIndicatorAdd	1237
ChartIndicatorDelete	1241
ChartIndicatorGet	1244
ChartIndicatorName	1246
ChartIndicatorsTotal	1247
ChartWindowOnDropped	1248
ChartPriceOnDropped	1249
ChartTimeOnDropped	1250
ChartXOnDropped	1251
ChartYOnDropped	1252
ChartSetSymbolPeriod	1253
ChartScreenShot	1254
16 Trade Functions	1257
OrderCalcMargin	1259
OrderCalcProfit	1260
OrderCheck	1261
OrderSend	1262
OrderSendAsync	1267
PositionsTotal	1278
PositionGetSymbol	1279
PositionSelect	1280
PositionSelectByTicket	1281
PositionGetDouble	1282
PositionGetInteger	1283
PositionGetString	1285
PositionGetTicket	1286
OrdersTotal	1287
OrderGetTicket	1288
OrderSelect	1290
OrderGetDouble	1291
OrderGetInteger	1292
OrderGetString	1293
HistorySelect	1294
HistorySelectByPosition	1296
HistoryOrderSelect	1297
HistoryOrdersTotal	1298
HistoryOrderGetTicket	1299
HistoryOrderGetDouble	1301
HistoryOrderGetInteger	1302
HistoryOrderGetString	1305
HistoryDealSelect	1306
HistoryDealsTotal	1307
HistoryDealGetTicket	1308
HistoryDealGetDouble	1310
HistoryDealGetInteger	1311

HistoryDealGetString	1314
17 Trade Signals.....	1315
SignalBaseGetDouble	1316
SignalBaseGetInteger	1317
SignalBaseGetString	1318
SignalBaseSelect	1319
SignalBaseTotal	1320
SignalInfoGetDouble	1321
SignalInfoGetInteger	1322
SignalInfoGetString	1323
SignalInfoSetDouble	1324
SignalInfoSetInteger	1325
SignalSubscribe	1326
SignalUnSubscribe	1327
18 Global Variables of the Terminal.....	1328
GlobalVariableCheck	1329
GlobalVariableTime	1330
GlobalVariableDel	1331
GlobalVariableGet	1332
GlobalVariableName	1333
GlobalVariableSet	1334
GlobalVariablesFlush	1335
GlobalVariableTemp	1336
GlobalVariableSetOnCondition	1337
GlobalVariablesDeleteAll	1338
GlobalVariablesTotal	1339
19 File Functions.....	1340
FileFindFirst	1343
FileFindNext	1345
FileFindClose	1347
FileIsExist	1349
FileOpen	1352
FileClose	1355
FileCopy	1356
FileDelete	1359
FileMove	1361
FileFlush	1363
FileGetInteger	1365
FileIsEnding	1368
FileIsLineEnding	1370
FileReadArray	1375
FileReadBool	1377
FileReadDatetime	1380
FileReadDouble	1383
FileReadFloat	1386
FileReadInteger	1389
FileReadLong	1393
FileReadNumber	1396
FileReadString	1401
FileReadStruct	1403
FileSeek	1407
FileSize	1410
FileTell	1412
FileWrite	1415
FileWriteArray	1418
FileWriteDouble	1421
FileWriteFloat	1424

FileWriteInteger	1426
FileWriteLong	1429
FileWriteString	1431
FileWriteStruct	1434
FileLoad	1437
FileSave	1439
FolderCreate	1441
FolderDelete	1444
FolderClean	1447
20 Custom Indicators	1450
Indicator Styles in Examples	1454
DRAW_NONE	1461
DRAW_LINE	1464
DRAW_SECTION	1468
DRAW_HISTOGRAM	1472
DRAW_HISTOGRAM2	1476
DRAW_ARROW	1480
DRAW_ZIGZAG	1485
DRAW_FILLING	1490
DRAW_BARS	1495
DRAW_CANDLES	1501
DRAW_COLOR_LINE	1507
DRAW_COLOR_SECTION	1512
DRAW_COLOR_HISTOGRAM	1518
DRAW_COLOR_HISTOGRAM2	1523
DRAW_COLOR_ARROW	1528
DRAW_COLOR_ZIGZAG	1534
DRAW_COLOR_BARS	1539
DRAW_COLOR_CANDLES	1546
Connection between Indicator Properties and Functions	1553
SetIndexBuffer	1556
IndicatorSetDouble	1559
IndicatorSetInteger	1563
IndicatorSetString	1567
PlotIndexSetDouble	1570
PlotIndexSetInteger	1571
PlotIndexSetString	1575
PlotIndexGetInteger	1576
21 Object Functions	1579
ObjectCreate	1581
ObjectName	1585
ObjectDelete	1586
ObjectsDeleteAll	1587
ObjectFind	1588
ObjectGetTimeByValue	1589
ObjectGetValueByTime	1590
ObjectMove	1591
ObjectsTotal	1592
ObjectSetDouble	1593
ObjectSetInteger	1596
ObjectSetString	1599
ObjectGetDouble	1601
ObjectGetInteger	1602
ObjectGetString	1603
TextSetFont	1605
TextOut	1607
TextGetSize	1611

22	Technical Indicators	1612
	iAC	1615
	iAD	1620
	iADX	1625
	iADXWilder	1630
	iAlligator	1635
	iAMA	1642
	iAO	1647
	iATR	1652
	iBearsPower	1657
	iBands	1662
	iBullsPower	1668
	iCCI	1673
	iChaikin	1678
	iCustom	1683
	iDEMA	1686
	iDeMarker	1691
	iEnvelopes	1696
	iForce	1702
	iFractals	1707
	iFrAMA	1712
	iGator	1717
	ilchimoku	1724
	iBWMFI	1731
	iMomentum	1736
	iMFI	1741
	iMA	1746
	iOsMA	1751
	iMACD	1756
	iOBV	1762
	iSAR	1767
	iRSI	1772
	iRVI	1777
	iStdDev	1782
	iStochastic	1787
	iTEMA	1793
	iTriX	1798
	iWPR	1803
	iVIDyA	1808
	iVolumes	1813
23	Working with Optimization Results.....	1818
	FrameFirst	1819
	FrameFilter	1820
	FrameNext	1821
	FrameInputs	1822
	FrameAdd	1823
	ParameterGetRange	1824
	ParameterSetRange	1827
24	Working with Events	1829
	EventSetMillisecondTimer	1830
	EventSetTimer	1831
	EventKillTimer	1832
	EventChartCustom	1833
25	Working with OpenCL.....	1839
	CLHandleType	1841
	CLGetInfoInteger	1842
	CLGetInfoString	1845

CLContextCreate	1848
CLContextFree	1849
CLGetDeviceInfo	1850
CLProgramCreate	1854
CLProgramFree	1859
CLKernelCreate	1860
CLKernelFree	1861
CLSetKernelArg	1862
CLSetKernelArgMem	1863
CLSetKernelArgMemLocal	1864
CLBufferCreate	1865
CLBufferFree	1866
CLBufferWrite	1867
CLBufferRead	1868
CLExecute	1869
CLExecutionStatus	1871
26 Standard Library	1872
Mathematics	1873
Statistics	1874
Statistical Characteristics	1877
MathMean	1878
MathVariance.....	1879
MathSkewness	1880
MathKurtosis.....	1881
MathMoments.....	1882
MathMedian.....	1883
MathStandardDeviation.....	1884
MathAverageDeviation.....	1885
Normal Distribution.....	1886
MathProbabilityDensityNormal.....	1890
MathCumulativeDistributionNormal.....	1892
MathQuantileNormal.....	1894
MathRandomNormal.....	1896
MathMomentsNormal.....	1897
Log-normal distribution	1898
MathProbabilityDensityLognormal.....	1902
MathCumulativeDistributionLognormal.....	1904
MathQuantileLognormal.....	1906
MathRandomLognormal.....	1908
MathMomentsLognormal.....	1909
Beta distribution	1910
MathProbabilityDensityBeta	1914
MathCumulativeDistributionBeta	1916
MathQuantileBeta.....	1918
MathRandomBeta.....	1920
MathMomentsBeta	1921
Noncentral beta distribution.....	1922
MathProbabilityDensityNoncentralBeta	1926
MathCumulativeDistributionNoncentralBeta.....	1928
MathQuantileNoncentralBeta.....	1930
MathRandomNoncentralBeta.....	1932
MathMomentsNoncentralBeta	1933
Gamma distribution.....	1934
MathProbabilityDensityGamma	1938
MathCumulativeDistributionGamma	1940
MathQuantileGamma.....	1942
MathRandomGamma.....	1944
MathMomentsGamma	1945

Chi-squared distribution.....	1946
MathProbabilityDensityChiSquare	1950
MathCumulativeDistributionChiSquare.....	1952
MathQuantileChiSquare.....	1954
MathRandomChiSquare.....	1956
MathMomentsChiSquare	1957
Noncentral chi-squared distribution.....	1958
MathProbabilityDensityNoncentralChiSquare.....	1962
MathCumulativeDistributionNoncentralChiSquare.....	1964
MathQuantileNoncentralChiSquare	1966
MathRandomNoncentralChiSquare	1968
MathMomentsNoncentralChiSquare.....	1969
Exponential distribution	1970
MathProbabilityDensityExponential.....	1974
MathCumulativeDistributionExponential.....	1976
MathQuantileExponential.....	1978
MathRandomExponential.....	1980
MathMomentsExponential.....	1981
F-distribution.....	1982
MathProbabilityDensityF.....	1986
MathCumulativeDistributionF.....	1988
MathQuantileF.....	1990
MathRandomF.....	1992
MathMomentsF.....	1993
Noncentral F-distribution.....	1994
MathProbabilityDensityNoncentralF.....	1998
MathCumulativeDistributionNoncentralF.....	2000
MathQuantileNoncentralF.....	2002
MathRandomNoncentralF.....	2004
MathMomentsNoncentralF.....	2005
T-distribution.....	2006
MathProbabilityDensityT.....	2010
MathCumulativeDistributionT.....	2012
MathQuantileT.....	2014
MathRandomT.....	2016
MathMomentsT.....	2017
Noncentral t-distribution.....	2018
MathProbabilityDensityNoncentralT.....	2022
MathCumulativeDistributionNoncentralT.....	2024
MathQuantileNoncentralT.....	2026
MathRandomNoncentralT.....	2028
MathMomentsNoncentralT.....	2029
Logistic distribution.....	2030
MathProbabilityDensityLogistic.....	2034
MathCumulativeDistributionLogistic	2036
MathQuantileLogistic	2038
MathRandomLogistic	2040
MathMomentsLogistic.....	2041
Cauchy distribution.....	2042
MathProbabilityDensityCauchy.....	2046
MathCumulativeDistributionCauchy.....	2048
MathQuantileCauchy	2050
MathRandomCauchy	2052
MathMomentsCauchy.....	2053
Uniform distribution	2054
MathProbabilityDensityUniform.....	2058
MathCumulativeDistributionUniform.....	2060
MathQuantileUniform.....	2062

MathRandomUniform.....	2064
MathMomentsUniform.....	2065
Weibull distribution.....	2066
MathProbabilityDensityWeibull.....	2070
MathCumulativeDistributionWeibull.....	2072
MathQuantileWeibull.....	2074
MathRandomWeibull.....	2076
MathMomentsWeibull.....	2077
Binomial distribution.....	2078
MathProbabilityDensityBinomial.....	2081
MathCumulativeDistributionBinomial.....	2083
MathQuantileBinomial.....	2085
MathRandomBinomial.....	2087
MathMomentsBinomial.....	2088
Negative binomial distribution.....	2089
MathProbabilityDensityNegativeBinomial.....	2092
MathCumulativeDistributionNegativeBinomial.....	2094
MathQuantileNegativeBinomial.....	2096
MathRandomNegativeBinomial.....	2098
MathMomentsNegativeBinomial.....	2099
Geometric distribution.....	2100
MathProbabilityDensityGeometric.....	2104
MathCumulativeDistributionGeometric.....	2106
MathQuantileGeometric.....	2108
MathRandomGeometric.....	2110
MathMomentsGeometric.....	2111
Hypergeometric distribution.....	2112
MathProbabilityDensityHypergeometric.....	2116
MathCumulativeDistributionHypergeometric.....	2118
MathQuantileHypergeometric.....	2120
MathRandomHypergeometric.....	2122
MathMomentsHypergeometric.....	2123
Poisson distribution.....	2124
MathProbabilityDensityPoisson.....	2128
MathCumulativeDistributionPoisson.....	2130
MathQuantilePoisson.....	2132
MathRandomPoisson.....	2134
MathMomentsPoisson.....	2135
Subfunctions.....	2136
MathRandomNonZero.....	2141
MathMoments.....	2142
MathPowInt.....	2143
MathFactorial.....	2144
MathTrunc.....	2145
MathRound.....	2146
MathArctan2.....	2148
MathGamma.....	2150
MathGammaLog.....	2151
MathBeta.....	2152
MathBetaLog.....	2153
MathBetaIncomplete.....	2154
MathGammaIncomplete.....	2155
MathBinomialCoefficient.....	2156
MathBinomialCoefficientLog.....	2157
MathHypergeometric2F2.....	2158
MathSequence.....	2159
MathSequenceByCount.....	2160
MathReplicate.....	2161

MathReverse.....	2162
MathIdentical.....	2163
MathUnique.....	2164
MathQuickSortAscending.....	2165
MathQuickSortDescending.....	2166
MathQuickSort.....	2167
MathOrder.....	2168
MathBitwiseNot.....	2169
MathBitwiseAnd.....	2170
MathBitwiseOr.....	2171
MathBitwiseXor.....	2172
MathBitwiseShiftL.....	2173
MathBitwiseShiftR.....	2174
MathCumulativeSum.....	2175
MathCumulativeProduct.....	2176
MathCumulativeMin.....	2177
MathCumulativeMax.....	2178
MathSin.....	2179
MathCos.....	2180
MathTan.....	2181
MathArcsin.....	2182
MathArccos.....	2183
MathArctan.....	2184
MathSinPi.....	2185
MathCosPi.....	2186
MathTanPi.....	2187
MathAbs.....	2188
MathCeil.....	2189
MathFloor.....	2190
MathSqrt.....	2191
MathExp.....	2192
MathPow.....	2193
MathLog.....	2194
MathLog2.....	2195
MathLog10.....	2196
MathLog1p.....	2197
MathDifference.....	2198
MathSample.....	2200
MathTukeySummary.....	2203
MathRange.....	2204
MathMin.....	2205
MathMax.....	2206
MathSum.....	2207
MathProduct.....	2208
MathStandardDeviation.....	2209
MathAverageDeviation.....	2210
MathMedian.....	2211
MathMean.....	2212
MathVariance.....	2213
MathSkewness.....	2214
MathKurtosis.....	2215
MathExpm1.....	2216
MathSinh.....	2217
MathCosh.....	2218
MathTanh.....	2219
MathArcsinh.....	2220
MathArccosh.....	2221
MathArctanh.....	2222

MathSignif.....	2223
MathRank	2225
MathCorrelationPearson.....	2226
MathCorrelationSpearman.....	2227
MathCorrelationKendall.....	2228
MathQuantile.....	2229
MathProbabilityDensityEmpirical.....	2230
MathCumulativeDistributionEmpirical.....	2231
Fuzzy Logic.....	2232
Membership functions.....	2233
CConstantMembershipFunction.....	2235
GetValue	2237
CCompositeMembershipFunction.....	2238
CompositionType.....	2240
MembershipFunctions.....	2240
GetValue	2240
CDifferencTwoSigmoidalMembershipFunction.....	2242
A1	2244
A2	2244
C1	2245
C2	2245
GetValue	2246
CGeneralizedBellShapedMembershipFunction.....	2247
A	2249
B	2249
C	2250
GetValue	2250
CNormalCombinationMembershipFunction.....	2251
B1	2253
B2	2253
Sigma1	2254
Sigma2	2254
GetValue	2255
CNormalMembershipFunction.....	2256
B	2258
Sigma	2258
GetValue	2259
CP_ShapedMembershipFunction.....	2260
A	2262
B	2262
C	2263
D	2263
GetValue	2263
CProductTwoSigmoidalMembershipFunctions.....	2265
A1	2267
A2	2267
C1	2268
C2	2268
GetValue	2269
CS_ShapedMembershipFunction.....	2270
A	2272
B	2272
GetValue	2273
CSigmoidalMembershipFunction.....	2274
A	2276
C	2276
GetValue	2277
CTrapezoidMembershipFunction.....	2278

X1	2280
X2	2280
X3	2281
X4	2281
GetValue	2282
CTriangularMembershipFunction	2283
X1	2285
X2	2285
X3	2286
ToNormalMF	2286
GetValue	2286
CZ_ShapedMembershipFunction	2288
A	2290
B	2290
GetValue	2291
IMembershipFunction	2292
GetValue	2292
Fuzzy systems rules	2293
CMamdaniFuzzyRule	2294
Conclusion	2295
Weight	2295
CSugenoFuzzyRule	2296
Conclusion	2296
CSingleCondition	2298
Not	2298
Term	2299
Var	2299
CConditions	2301
ConditionsList	2301
Not	2302
Op	2302
CGenericFuzzyRule	2303
Conclusion	2303
Condition	2304
CreateCondition	2304
Fuzzy systems variables	2306
CFuzzyVariable	2307
AddTerm	2308
GetTermByName	2308
Max	2308
Min	2309
Terms	2309
Values	2309
CSugenoVariable	2311
Functions	2311
GetFuncByName	2312
Values	2312
Fuzzy terms	2313
MembershipFunction	2314
Fuzzy systems	2315
Mamdani system	2316
AggregationMethod	2316
Calculate	2317
DefuzzificationMethod	2317
EmptyRule	2317
ImplicationMethod	2317
Output	2318
OutputByName	2318

ParseRule	2318
Rules	2318
Sugeno system.....	2320
Calculate	2320
CreateSugenoFunction.....	2321
EmptyRule	2322
Output	2322
OutputByName	2322
ParseRule	2322
Rules	2323
OpenCL	2324
BufferCreate.....	2326
BufferFree.....	2327
BufferFromArray.....	2328
BufferRead	2329
BufferWrite.....	2330
Execute.....	2331
GetContext	2332
GetKernel	2333
GetKernelName.....	2334
GetProgram.....	2335
Initialize.....	2336
KernelCreate.....	2337
KernelFree.....	2338
SetArgument	2339
SetArgumentBuffer	2340
SetArgumentLocalMemory.....	2341
SetBuffersCount.....	2342
SetKernelsCount.....	2343
Shutdown	2344
SupportDouble.....	2345
Basic Class CObject	2346
Prev	2347
Prev	2348
Next	2349
Next	2350
Compare.....	2351
Save	2353
Load	2355
Type	2357
Data Collections	2358
CArray	2359
Step	2361
Step	2362
Total	2363
Available	2364
Max	2365
IsSorted	2366
SortMode	2367
Clear	2368
Sort	2369
Save	2370
Load	2371
CArrayChar	2372
Reserve	2375
Resize	2376
Shutdown	2377
Add	2378

AddArray	2379
AddArray	2380
Insert	2382
InsertArray.....	2383
InsertArray.....	2384
AssignArray.....	2386
AssignArray.....	2387
Update	2389
Shift	2390
Delete	2391
DeleteRange.....	2392
At	2393
CompareArray	2395
CompareArray	2396
InsertSort	2397
Search	2398
SearchGreat.....	2399
SearchLess	2400
SearchGreatOrEqual.....	2401
SearchLessOrEqual.....	2402
SearchFirst.....	2403
SearchLast	2404
SearchLinear.....	2405
Save	2406
Load	2407
Type	2409
CArrayShort.....	2410
Reserve	2413
Resize	2414
Shutdown	2415
Add	2416
AddArray	2417
AddArray	2418
Insert	2420
InsertArray.....	2421
InsertArray.....	2422
AssignArray.....	2424
AssignArray.....	2425
Update	2427
Shift	2428
Delete	2429
DeleteRange.....	2430
At	2431
CompareArray	2433
CompareArray	2434
InsertSort	2435
Search	2436
SearchGreat.....	2437
SearchLess	2438
SearchGreatOrEqual.....	2439
SearchLessOrEqual.....	2440
SearchFirst.....	2441
SearchLast	2442
SearchLinear.....	2443
Save	2444
Load	2446
Type	2448
CArrayInt.....	2449

Reserve	2452
Resize	2453
Shutdown	2454
Add	2455
AddArray	2456
AddArray	2457
Insert	2459
InsertArray.....	2460
InsertArray.....	2461
AssignArray.....	2463
AssignArray.....	2464
Update	2466
Shift	2467
Delete	2468
DeleteRange.....	2469
At	2470
CompareArray	2472
CompareArray	2473
InsertSort	2474
Search	2475
SearchGreat.....	2476
SearchLess	2477
SearchGreatOrEqual.....	2478
SearchLessOrEqual.....	2479
SearchFirst.....	2480
SearchLast	2481
SearchLinear.....	2482
Save	2483
Load	2485
Type	2487
CArrayLong	2488
Reserve	2491
Resize	2492
Shutdown	2493
Add	2494
AddArray	2495
AddArray	2496
Insert	2498
InsertArray.....	2499
InsertArray.....	2500
AssignArray.....	2502
AssignArray.....	2503
Update	2505
Shift	2506
Delete	2507
DeleteRange.....	2508
At	2509
CompareArray	2511
CompareArray	2512
InsertSort	2513
Search	2514
SearchGreat.....	2515
SearchLess	2516
SearchGreatOrEqual.....	2517
SearchLessOrEqual.....	2518
SearchFirst.....	2519
SearchLast	2520
SearchLinear.....	2521

Save	2522
Load	2524
Type	2526
CArrayFloat	2527
Delta	2530
Reserve	2531
Resize	2532
Shutdown	2533
Add	2534
AddArray	2535
AddArray	2536
Insert	2538
InsertArray.....	2539
InsertArray.....	2540
AssignArray.....	2542
AssignArray.....	2543
Update	2545
Shift	2546
Delete	2547
DeleteRange.....	2548
At	2549
CompareArray	2551
CompareArray	2552
InsertSort	2553
Search	2554
SearchGreat.....	2555
SearchLess	2556
SearchGreatOrEqual.....	2557
SearchLessOrEqual.....	2558
SearchFirst.....	2559
SearchLast	2560
SearchLinear.....	2561
Save	2562
Load	2564
Type	2566
CArrayDouble.....	2567
Delta	2570
Reserve	2571
Resize	2572
Shutdown	2573
Add	2574
AddArray	2575
AddArray	2576
Insert	2578
InsertArray.....	2579
InsertArray.....	2580
AssignArray.....	2582
AssignArray.....	2583
Update	2585
Shift	2586
Delete	2587
DeleteRange.....	2588
At	2589
CompareArray	2591
CompareArray	2592
Minimum	2593
Maximum	2594
InsertSort	2595

Search	2596
SearchGreat.....	2597
SearchLess	2598
SearchGreatOrEqual.....	2599
SearchLessOrEqual.....	2600
SearchFirst.....	2601
SearchLast	2602
SearchLinear.....	2603
Save	2604
Load	2606
Type	2608
CArrayString.....	2609
Reserve	2612
Resize	2613
Shutdown	2614
Add	2615
AddArray	2616
AddArray	2617
Insert	2619
InsertArray.....	2620
InsertArray.....	2621
AssignArray.....	2623
AssignArray.....	2624
Update	2626
Shift	2627
Delete	2628
DeleteRange.....	2629
At	2630
CompareArray.....	2632
CompareArray.....	2633
InsertSort	2634
Search	2635
SearchGreat.....	2636
SearchLess	2637
SearchGreatOrEqual.....	2638
SearchLessOrEqual.....	2639
SearchFirst.....	2640
SearchLast	2641
SearchLinear.....	2642
Save	2643
Load	2645
Type	2647
CArrayObj.....	2648
FreeMode	2653
FreeMode	2654
Reserve	2656
Resize	2657
Clear	2658
Shutdown	2659
CreateElement.....	2660
Add	2662
AddArray	2663
Insert	2666
InsertArray.....	2668
AssignArray.....	2670
Update	2672
Shift	2673
Detach	2674

Delete	2675
DeleteRange	2676
At	2677
CompareArray	2678
InsertSort	2679
Search	2680
SearchGreat	2681
SearchLess	2682
SearchGreatOrEqual	2683
SearchLessOrEqual	2685
SearchFirst	2686
SearchLast	2687
Save	2688
Load	2689
Type	2691
CList	2692
FreeMode	2694
FreeMode	2695
Total	2697
IsSorted	2698
SortMode	2699
CreateElement	2700
Add	2701
Insert	2702
DetachCurrent	2704
DeleteCurrent	2705
Delete	2706
Clear	2707
IndexOf	2708
GetNodeAtIndex	2709
GetFirstNode	2710
GetPrevNode	2711
GetCurrentNode	2712
GetNextNode	2713
GetLastNode	2714
Sort	2715
MoveToIndex	2716
Exchange	2717
CompareList	2718
Search	2719
Save	2720
Load	2722
Type	2724
CTreeNode	2725
Owner	2730
Left	2731
Right	2732
Balance	2733
BalanceL	2734
BalanceR	2735
CreateSample	2736
RefreshBalance	2737
GetNext	2738
SaveNode	2739
LoadNode	2740
Type	2741
CTree	2742
Root	2748

CreateElement.....	2749
Insert	2750
Detach	2751
Delete	2752
Clear	2753
Find	2754
Save	2755
Load	2756
Type	2757
Files	2758
CFile	2759
Handle	2761
Filename	2762
Flags	2763
SetUnicode.....	2764
SetCommon.....	2765
Open	2766
Close	2767
Delete	2768
IsExist	2769
Copy	2770
Move	2771
Size	2772
Tell	2773
Seek	2774
Flush	2775
IsEnding	2776
IsLineEnding.....	2777
FolderCreate.....	2778
FolderDelete.....	2779
FolderClean.....	2780
FileFindFirst.....	2781
FileFindNext.....	2782
FileFindClose	2783
CFileBin	2784
Open	2786
WriteChar	2787
WriteShort	2788
WriteInteger.....	2789
WriteLong	2790
WriteFloat.....	2791
WriteDouble.....	2792
WriteString.....	2793
WriteCharArray	2794
WriteShortArray	2795
WriteIntegerArray.....	2796
WriteLongArray	2797
WriteFloatArray.....	2798
WriteDoubleArray.....	2799
WriteObject	2800
ReadChar	2801
ReadShort	2802
ReadInteger.....	2803
ReadLong	2804
ReadFloat	2805
ReadDouble.....	2806
ReadString.....	2807
ReadCharArray	2808

ReadShortArray.....	2809
ReadIntegerArray.....	2810
ReadLongArray.....	2811
ReadFloatArray.....	2812
ReadDoubleArray.....	2813
ReadObject.....	2814
CFileTxt.....	2815
Open.....	2816
WriteString.....	2817
ReadString.....	2818
Strings.....	2819
CString.....	2820
Str.....	2822
Len.....	2823
Copy.....	2824
Fill.....	2825
Assign.....	2826
Append.....	2827
Insert.....	2828
Compare.....	2829
CompareNoCase.....	2830
Left.....	2831
Right.....	2832
Mid.....	2833
Trim.....	2834
TrimLeft.....	2835
TrimRight.....	2836
Clear.....	2837
ToUpper.....	2838
ToLower.....	2839
Reverse.....	2840
Find.....	2841
FindRev.....	2842
Remove.....	2843
Replace.....	2844
Graphic Objects.....	2845
CChartObject.....	2846
ChartId.....	2849
Window.....	2850
Name.....	2851
NumPoints.....	2852
Attach.....	2853
SetPoint.....	2854
Delete.....	2855
Detach.....	2856
ShiftObject.....	2857
ShiftPoint.....	2858
Time.....	2859
Price.....	2861
Color.....	2863
Style.....	2864
Width.....	2865
Background.....	2866
Selected.....	2867
Selectable.....	2868
Description.....	2869
Tooltip.....	2870
Timeframes.....	2871

Z_Order	2872
CreateTime.....	2873
LevelsCount	2874
LevelColor	2875
LevelStyle	2877
LevelWidth	2879
LevelValue	2881
LevelDescription	2883
GetInteger	2885
SetInteger	2887
GetDouble	2889
SetDouble	2891
GetString	2893
SetString	2895
Save	2897
Load	2898
Type	2899
Line Objects.....	2900
CChartObjectVLine.....	2901
Create	2902
Type	2903
CChartObjectHLine.....	2904
Create	2905
Type	2906
CChartObjectTrend	2907
Create	2909
RayLeft	2910
RayRight	2911
Save	2912
Load	2913
Type	2914
CChartObjectTrendByAngle.....	2915
Create	2917
Angle	2918
Type	2919
CChartObjectCycles.....	2920
Create	2921
Type	2922
Channel Objects.....	2923
CChartObjectChannel.....	2924
Create	2926
Type	2927
CChartObjectRegression.....	2928
Create	2930
Type	2931
CChartObjectStdDevChannel.....	2932
Create	2934
Deviations.....	2935
Save	2936
Load	2937
Type	2938
CChartObjectPitchfork.....	2939
Create	2941
Type	2942
Gann Tools.....	2943
CChartObjectGannLine.....	2944
Create	2946
PipsPerBar.....	2947

Save	2948
Load	2949
Type	2950
CChartObjectGannFan.....	2951
Create	2953
PipsPerBar.....	2954
Downtrend.....	2955
Save	2956
Load	2957
Type	2958
CChartObjectGannGrid.....	2959
Create	2961
PipsPerBar.....	2962
Downtrend.....	2963
Save	2964
Load	2965
Type	2966
Fibonacci Tools.....	2967
CChartObjectFibo	2968
Create	2970
Type	2971
CChartObjectFiboTimes	2972
Create	2973
Type	2974
CChartObjectFiboFan.....	2975
Create	2976
Type	2977
CChartObjectFiboArc.....	2978
Create	2980
Scale	2981
Ellipse	2982
Save	2983
Load	2984
Type	2985
CChartObjectFiboChannel.....	2986
Create	2988
Type	2989
CChartObjectFiboExpansion.....	2990
Create	2992
Type	2993
Elliott Tools.....	2994
CChartObjectElliottWave3.....	2995
Create	2997
Degree	2998
Lines	2999
Save	3000
Load	3001
Type	3002
CChartObjectElliottWave5.....	3003
Create	3005
Type	3007
Shape Objects.....	3008
CChartObjectRectangle.....	3009
Create	3010
Type	3011
CChartObjectTriangle.....	3012
Create	3013
Type	3014

CChartObjectEllipse.....	3015
Create	3016
Type	3017
Arrow Objects.....	3018
CChartObjectArrow.....	3019
Create	3021
ArrowCode.....	3023
Anchor	3025
Save	3027
Load	3028
Type	3029
Arrows with fixed code.....	3030
Create	3032
ArrowCode.....	3034
Type	3035
Control Objects.....	3036
CChartObjectText.....	3037
Create	3039
Angle	3040
Font	3041
FontSize	3042
Anchor	3043
Save	3044
Load	3045
Type	3046
CChartObjectLabel.....	3047
Create	3049
X_Distance.....	3050
Y_Distance.....	3051
X_Size	3052
Y_Size	3053
Corner	3054
Time	3055
Price	3056
Save	3057
Load	3058
Type	3059
CChartObjectEdit.....	3060
Create	3062
TextAlign	3063
X_Size	3064
Y_Size	3065
BackColor	3066
BorderColor	3067
ReadOnly	3068
Angle	3069
Save	3070
Load	3071
Type	3072
CChartObjectButton.....	3073
State	3075
Save	3076
Load	3077
Type	3078
CChartObjectSubChart.....	3079
Create	3081
X_Distance.....	3082
Y_Distance.....	3083

Corner	3084
X_Size	3085
Y_Size	3086
Symbol	3087
Period	3088
Scale	3089
DateScale	3090
PriceScale	3091
Time	3092
Price	3093
Save	3094
Load	3095
Type	3096
CChartObjectBitmap	3097
Create	3099
BmpFile	3100
X_Offset	3101
Y_Offset	3102
Save	3103
Load	3104
Type	3105
CChartObjectBmpLabel	3106
Create	3108
X_Distance	3109
Y_Distance	3110
X_Offset	3111
Y_Offset	3112
Corner	3113
X_Size	3114
Y_Size	3115
BmpFileOn	3116
BmpFileOff	3117
State	3118
Time	3119
Price	3120
Save	3121
Load	3122
Type	3123
CChartObjectRectLabel	3124
Create	3126
X_Size	3127
Y_Size	3128
BackColor	3129
Angle	3130
BorderType	3131
Save	3132
Load	3133
Type	3134
Custom Graphics	3135
Attach	3139
Arc	3140
Pie	3144
FillPolygon	3148
FillEllipse	3149
GetDefaultColor	3150
ChartObjectName	3151
Circle	3152
CircleAA	3153

Create	3154
CircleWu.....	3155
CreateBitmap.....	3156
CreateBitmapLabel.....	3158
Destroy	3160
Ellipse	3161
Erase	3162
EllipseAA	3163
EllipseWu.....	3164
Fill	3165
FillCircle.....	3166
FillRectangle.....	3167
FillTriangle.....	3168
FontAngleGet	3169
FontAngleSet.....	3170
FontFlagsGet.....	3171
FontFlagsSet.....	3172
FontGet.....	3173
FontNameGet.....	3174
FontNameSet.....	3175
FontSet	3176
FontSizeGet.....	3177
FontSizeSet	3178
Height	3179
Line	3180
LineAA	3181
LineWu.....	3182
LineHorizontal.....	3183
LineStyleSet.....	3184
LineVertical.....	3185
LineThickHorizontal.....	3186
LineThickVertical.....	3187
LineThick	3188
LoadFromFile.....	3189
PixelGet.....	3190
PixelSet	3191
PixelSetAA	3192
Polygon.....	3193
PolygonAA	3194
PolygonWu.....	3195
PolygonThick.....	3196
PolygonSmooth.....	3197
Polyline.....	3198
PolylineAA.....	3199
PolylineWu.....	3200
PolylineSmooth.....	3201
PolylineThick.....	3202
Rectangle.....	3203
Resize	3204
ResourceName.....	3205
TextHeight.....	3206
TextOut.....	3207
TextSize	3208
TextWidth.....	3209
TransparentLevelSet.....	3210
Triangle.....	3211
TriangleAA.....	3212
TriangleWu.....	3213

Update.....	3214
Width	3215
Price Charts	3216
ChartID.....	3221
Mode	3222
Foreground.....	3223
Shift	3224
ShiftSize.....	3225
AutoScroll.....	3226
Scale	3227
ScaleFix.....	3228
ScaleFix_11.....	3229
FixedMax	3230
FixedMin.....	3231
PointsPerBar.....	3232
ScalePPB.....	3233
ShowOHLC.....	3234
ShowLineBid.....	3235
ShowLineAsk	3236
ShowLastLine.....	3237
ShowPeriodSep.....	3238
ShowGrid.....	3239
ShowVolumes.....	3240
ShowObjectDescr.....	3241
ShowDateScale.....	3242
ShowPriceScale.....	3243
ColorBackground.....	3244
ColorForeground.....	3245
ColorGrid	3246
ColorBarUp.....	3247
ColorBarDown.....	3248
ColorCandleBull.....	3249
ColorCandleBear.....	3250
ColorChartLine.....	3251
ColorVolumes.....	3252
ColorLineBid.....	3253
ColorLineAsk	3254
ColorLineLast.....	3255
ColorStopLevels	3256
VisibleBars.....	3257
WindowsTotal.....	3258
WindowsVisible.....	3259
WindowHandle.....	3260
FirstVisibleBar.....	3261
WidthInBars.....	3262
WidthInPixels	3263
HeightInPixels	3264
PriceMin	3265
PriceMax.....	3266
Attach	3267
FirstChart.....	3268
NextChart.....	3269
Open	3270
Detach.....	3271
Close	3272
BringToTop.....	3273
EventObjectCreate.....	3274
EventObjectDelete.....	3275

IndicatorAdd.....	3276
IndicatorDelete.....	3277
IndicatorsTotal.....	3278
IndicatorName.....	3279
Navigate.....	3280
Symbol.....	3281
Period.....	3282
Redraw.....	3283
GetInteger.....	3284
SetInteger.....	3285
GetDouble.....	3286
SetDouble.....	3287
GetString.....	3288
SetString.....	3289
SetSymbolPeriod.....	3290
ApplyTemplate.....	3291
ScreenShot.....	3292
WindowOnDropped.....	3293
PriceOnDropped.....	3294
TimeOnDropped.....	3295
XOnDropped.....	3296
YOnDropped.....	3297
Save.....	3298
Load.....	3299
Type.....	3300
Scientific Charts.....	3301
GraphPlot.....	3302
CAxis.....	3306
AutoScale.....	3308
Min.....	3309
Max.....	3310
Step.....	3311
Name.....	3312
Color.....	3313
ValuesSize.....	3314
ValuesWidth.....	3315
ValuesFormat.....	3316
ValuesDateTimeMode.....	3317
ValuesFunctionFormat.....	3318
ValuesFunctionFormatCBData.....	3320
NameSize.....	3321
ZeroLever.....	3322
DefaultStep.....	3323
MaxLabels.....	3324
MinGrace.....	3325
MaxGrace.....	3326
SelectAxisScale.....	3327
CColorGenerator.....	3328
Next.....	3329
Reset.....	3330
CCurve.....	3331
Type.....	3333
Name.....	3334
Color.....	3335
XMax.....	3336
XMin.....	3337
YMax.....	3338
YMin.....	3339

Size	3340
PointsSize	3341
PointsFill	3342
PointsColor	3343
GetX	3344
GetY	3345
LineStyle	3346
LinesIsSmooth	3347
LinesSmoothTension	3348
LinesSmoothStep	3349
LinesEndStyle	3350
LinesWidth	3351
HistogramWidth	3353
CustomPlotCBData	3354
CustomPlotFunction	3355
PointsType	3359
StepsDimension	3360
TrendLineCoefficients	3361
TrendLineColor	3362
TrendLineVisible	3363
Update	3365
Visible	3367
CGraphic	3368
Create	3371
Destroy	3372
Update	3373
ChartObjectName	3374
ResourceName	3375
XAxis	3376
YAxis	3377
GapSize	3378
BackgroundColor	3379
BackgroundMain	3380
BackgroundMainSize	3381
BackgroundMainColor	3382
BackgroundSub	3383
BackgroundSubSize	3384
BackgroundSubColor	3385
GridLineColor	3386
GridBackgroundColor	3387
GridCircleRadius	3388
GridCircleColor	3389
GridHasCircle	3390
GridAxisLineColor	3391
HistoryNameWidth	3392
HistoryNameSize	3393
HistorySymbolSize	3394
TextAdd	3395
LineAdd	3396
CurveAdd	3397
CurvePlot	3400
CurvePlotAll	3401
CurveTotal	3402
CurveGetByIndex	3403
CurveGetByName	3404
CurveRemoveByIndex	3405
CurveRemoveByName	3406
CurvesTotal	3407

MarksToAxisAdd	3408
MajorMarkSize.....	3409
FontSet	3410
FontGet	3411
Attach	3412
CalculateMaxMinValues.....	3413
Height	3414
IndentDown	3415
IndentLeft.....	3416
IndentRight.....	3417
IndentUp	3418
Redraw	3419
ResetParameters.....	3420
ScaleX	3421
ScaleY	3422
SetDefaultParameters.....	3423
Width	3424
Indicators	3425
Base classes.....	3426
CSpreadBuffer	3427
Size	3429
SetSymbolPeriod.....	3430
At	3431
Refresh	3432
RefreshCurrent.....	3433
CTimeBuffer	3434
Size	3436
SetSymbolPeriod.....	3437
At	3438
Refresh	3439
RefreshCurrent.....	3440
CTickVolumeBuffer	3441
Size	3443
SetSymbolPeriod.....	3444
At	3445
Refresh	3446
RefreshCurrent.....	3447
CRealVolumeBuffer	3448
Size	3450
SetSymbolPeriod.....	3451
At	3452
Refresh	3453
RefreshCurrent.....	3454
CDoubleBuffer.....	3455
Size	3457
SetSymbolPeriod.....	3458
At	3459
Refresh	3460
RefreshCurrent.....	3461
COpenBuffer	3462
Refresh	3463
RefreshCurrent.....	3464
CHighBuffer	3465
Refresh	3466
RefreshCurrent.....	3467
CLowBuffer.....	3468
Refresh	3469
RefreshCurrent.....	3470

CCloseBuffer	3471
Refresh	3472
RefreshCurrent.....	3473
CIndicatorBuffer	3474
Offset	3476
Name	3477
At	3478
Refresh	3479
RefreshCurrent.....	3480
CSeries	3481
Name	3483
BuffersTotal.....	3484
Timeframe.....	3485
Symbol	3486
Period	3487
RefreshCurrent.....	3488
BufferSize	3489
BufferResize	3490
Refresh	3491
PeriodDescription.....	3492
CPriceSeries.....	3493
BufferResize	3495
GetData	3496
Refresh	3497
MinIndex	3498
MinValue	3499
MaxIndex	3500
MaxValue	3501
CIndicator.....	3502
Handle	3504
Status	3505
FullRelease.....	3506
Create	3507
BufferResize	3508
BarsCalculated.....	3509
GetData	3510
Refresh	3513
Minimum	3514
MinValue	3515
Maximum	3516
MaxValue	3517
MethodDescription.....	3518
PriceDescription.....	3519
VolumeDescription.....	3520
AddToChart	3521
DeleteFromChart.....	3522
CIndicators.....	3523
Create	3524
Refresh	3525
Timeseries classes	3526
CiSpread	3527
Create	3529
BufferResize.....	3530
GetData	3531
Refresh	3533
CiTime	3534
Create	3536
BufferResize	3537

GetData	3538
Refresh	3540
CiTickVolume.....	3541
Create	3543
BufferResize.....	3544
GetData	3545
Refresh	3547
CiRealVolume.....	3548
Create	3550
BufferResize.....	3551
GetData	3552
Refresh	3554
CiOpen	3555
Create	3557
GetData	3558
CiHigh	3560
Create	3562
GetData	3563
CiLow	3565
Create	3567
GetData	3568
CiClose	3570
Create	3572
GetData	3573
Trend Indicators.....	3575
CiADX	3576
MaPeriod	3578
Create	3579
Main	3580
Plus	3581
Minus	3582
Type	3583
CiADXWilder.....	3584
MaPeriod	3586
Create	3587
Main	3588
Plus	3589
Minus	3590
Type	3591
CiBands	3592
MaPeriod	3594
MaShift	3595
Deviation	3596
Applied	3597
Create	3598
Base	3599
Upper	3600
Lower	3601
Type	3602
CiEnvelopes.....	3603
MaPeriod	3605
MaShift	3606
MaMethod.....	3607
Deviation	3608
Applied	3609
Create	3610
Upper	3611
Lower	3612

Type	3613
Cilchimoku.....	3614
TenkanSenPeriod.....	3616
KijunSenPeriod.....	3617
SenkouSpanBPeriod.....	3618
Create	3619
TenkanSen.....	3620
KijunSen	3621
SenkouSpanA.....	3622
SenkouSpanB.....	3623
ChinkouSpan.....	3624
Type	3625
CiMA	3626
MaPeriod	3628
MaShift	3629
MaMethod.....	3630
Applied	3631
Create	3632
Main	3633
Type	3634
CiSAR	3635
SarStep	3637
Maximum	3638
Create	3639
Main	3640
Type	3641
CiStdDev	3642
MaPeriod	3644
MaShift	3645
MaMethod.....	3646
Applied	3647
Create	3648
Main	3649
Type	3650
CiDEMA	3651
MaPeriod	3653
IndShift	3654
Applied	3655
Create	3656
Main	3657
Type	3658
CiTEMA	3659
MaPeriod	3661
IndShift	3662
Applied	3663
Create	3664
Main	3665
Type	3666
CiFrAMA	3667
MaPeriod	3669
IndShift	3670
Applied	3671
Create	3672
Main	3673
Type	3674
CiAMA	3675
MaPeriod	3677
FastEmaPeriod.....	3678

SlowEmaPeriod.....	3679
IndShift	3680
Applied	3681
Create	3682
Main	3683
Type	3684
CiVIDyA	3685
CmoPeriod.....	3687
EmaPeriod.....	3688
IndShift	3689
Applied	3690
Create	3691
Main	3692
Type	3693
Oscillators.....	3694
CiATR	3695
MaPeriod	3697
Create	3698
Main	3699
Type	3700
CiBearsPower.....	3701
MaPeriod	3703
Create	3704
Main	3705
Type	3706
CiBullsPower.....	3707
MaPeriod	3709
Create	3710
Main	3711
Type	3712
CiCCI	3713
MaPeriod	3715
Applied	3716
Create	3717
Main	3718
Type	3719
CiChaikin	3720
FastMaPeriod.....	3722
SlowMaPeriod.....	3723
MaMethod.....	3724
Applied	3725
Create	3726
Main	3727
Type	3728
CiDeMarker.....	3729
MaPeriod	3731
Create	3732
Main	3733
Type	3734
CiForce	3735
MaPeriod	3737
MaMethod.....	3738
Applied	3739
Create	3740
Main	3741
Type	3742
CiMACD	3743
FastEmaPeriod.....	3745

SlowEmaPeriod.....	3746
SignalPeriod.....	3747
Applied	3748
Create	3749
Main	3750
Signal	3751
Type	3752
CiMomentum.....	3753
MaPeriod	3755
Applied	3756
Create	3757
Main	3758
Type	3759
CiOsMA	3760
FastEmaPeriod.....	3762
SlowEmaPeriod.....	3763
SignalPeriod.....	3764
Applied	3765
Create	3766
Main	3767
Type	3768
CiRSI	3769
MaPeriod	3771
Applied	3772
Create	3773
Main	3774
Type	3775
CiRVI	3776
MaPeriod	3778
Create	3779
Main	3780
Signal	3781
Type	3782
CiStochastic.....	3783
Kperiod	3785
Dperiod	3786
Slowing	3787
MaMethod.....	3788
PriceField	3789
Create	3790
Main	3791
Signal	3792
Type	3793
CiTriX	3794
MaPeriod	3796
Applied	3797
Create	3798
Main	3799
Type	3800
CiWPR	3801
CalcPeriod.....	3803
Create	3804
Main	3805
Type	3806
Volume Indicators.....	3807
CiAD	3808
Applied	3810
Create	3811

Main	3812
Type	3813
CiMFI	3814
MaPeriod	3816
Applied	3817
Create	3818
Main	3819
Type	3820
CiOBV	3821
Applied	3823
Create	3824
Main	3825
Type	3826
CiVolumes	3827
Applied	3829
Create	3830
Main	3831
Type	3832
Bill Williams Indicators.....	3833
CiAC	3834
Create	3836
Main	3837
Type	3838
CiAlligator	3839
JawPeriod.....	3841
JawShift	3842
TeethPeriod.....	3843
TeethShift.....	3844
LipsPeriod.....	3845
LipsShift	3846
MaMethod.....	3847
Applied	3848
Create	3849
Jaw	3850
Teeth	3851
Lips	3852
Type	3853
CiAO	3854
Create	3856
Main	3857
Type	3858
CiFractals	3859
Create	3861
Upper	3862
Lower	3863
Type	3864
CiGator	3865
JawPeriod.....	3867
JawShift	3868
TeethPeriod.....	3869
TeethShift.....	3870
LipsPeriod.....	3871
LipsShift	3872
MaMethod.....	3873
Applied	3874
Create	3875
Upper	3876
Lower	3877

Type	3878
CiBWMFI	3879
Applied	3881
Create	3882
Main	3883
Type	3884
Custom indicators	3885
NumBuffers	3886
NumParams	3887
ParamType	3888
ParamLong	3889
ParamDouble	3890
ParamString	3891
Type	3892
Trade Classes	3893
CAccountInfo	3894
Login	3896
TradeMode	3897
TradeModeDescription	3898
Leverage	3899
StopoutMode	3900
StopoutModeDescription	3901
MarginMode	3902
MarginModeDescription	3903
TradeAllowed	3904
TradeExpert	3905
LimitOrders	3906
Balance	3907
Credit	3908
Profit	3909
Equity	3910
Margin	3911
FreeMargin	3912
MarginLevel	3913
MarginCall	3914
MarginStopOut	3915
Name	3916
Server	3917
Currency	3918
Company	3919
InfoInteger	3920
InfoDouble	3921
InfoString	3922
OrderProfitCheck	3923
MarginCheck	3924
FreeMarginCheck	3925
MaxLotCheck	3926
CSymbolInfo	3927
Refresh	3931
RefreshRates	3932
Name	3933
Select	3934
IsSynchronized	3935
Volume	3936
VolumeHigh	3937
VolumeLow	3938
Time	3939
Spread	3940

SpreadFloat.....	3941
TicksBookDepth.....	3942
StopsLevel.....	3943
FreezeLevel.....	3944
Bid.....	3945
BidHigh.....	3946
BidLow.....	3947
Ask.....	3948
AskHigh.....	3949
AskLow.....	3950
Last.....	3951
LastHigh.....	3952
LastLow.....	3953
TradeCalcMode.....	3954
TradeCalcModeDescription.....	3955
TradeMode.....	3956
TradeModeDescription.....	3957
TradeExecution.....	3958
TradeExecutionDescription.....	3959
SwapMode.....	3960
SwapModeDescription.....	3961
SwapRollover3days.....	3962
SwapRollover3daysDescription.....	3963
MarginInitial.....	3964
MarginMaintenance.....	3965
MarginLong.....	3966
MarginShort.....	3967
MarginLimit.....	3968
MarginStop.....	3969
MarginStopLimit.....	3970
TradeTimeFlags.....	3971
TradeFillFlags.....	3972
Digits.....	3973
Point.....	3974
TickValue.....	3975
TickValueProfit.....	3976
TickValueLoss.....	3977
TickSize.....	3978
ContractSize.....	3979
LotsMin.....	3980
LotsMax.....	3981
LotsStep.....	3982
LotsLimit.....	3983
SwapLong.....	3984
SwapShort.....	3985
CurrencyBase.....	3986
CurrencyProfit.....	3987
CurrencyMargin.....	3988
Bank.....	3989
Description.....	3990
Path.....	3991
SessionDeals.....	3992
SessionBuyOrders.....	3993
SessionSellOrders.....	3994
SessionTurnover.....	3995
SessionInterest.....	3996
SessionBuyOrdersVolume.....	3997
SessionSellOrdersVolume.....	3998

SessionOpen.....	3999
SessionClose.....	4000
SessionAW	4001
SessionPriceSettlement	4002
SessionPriceLimitMin	4003
SessionPriceLimitMax.....	4004
InfoInteger.....	4005
InfoDouble.....	4006
InfoString	4007
NormalizePrice.....	4008
COrderInfo.....	4009
Ticket	4011
TimeSetup	4012
TimeSetupMsc.....	4013
OrderType.....	4014
TypeDescription.....	4015
State	4016
StateDescription.....	4017
TimeExpiration.....	4018
TimeDone	4019
TimeDoneMsc.....	4020
TypeFilling	4021
TypeFillingDescription.....	4022
TypeTime	4023
TypeTimeDescription.....	4024
Magic	4025
PositionId	4026
VolumeInitial.....	4027
VolumeCurrent.....	4028
PriceOpen	4029
StopLoss	4030
TakeProfit	4031
PriceCurrent.....	4032
PriceStopLimit	4033
Symbol	4034
Comment	4035
InfoInteger.....	4036
InfoDouble.....	4037
InfoString	4038
StoreState.....	4039
CheckState.....	4040
Select	4041
SelectByIndex.....	4042
CHistoryOrderInfo.....	4043
TimeSetup	4045
TimeSetupMsc.....	4046
OrderType.....	4047
TypeDescription.....	4048
State	4049
StateDescription.....	4050
TimeExpiration.....	4051
TimeDone	4052
TimeDoneMsc.....	4053
TypeFilling	4054
TypeFillingDescription.....	4055
TypeTime	4056
TypeTimeDescription.....	4057
Magic	4058

PositionId	4059
VolumeInitial.....	4060
VolumeCurrent.....	4061
PriceOpen	4062
StopLoss	4063
TakeProfit	4064
PriceCurrent.....	4065
PriceStopLimit	4066
Symbol	4067
Comment	4068
InfoInteger.....	4069
InfoDouble.....	4070
InfoString	4071
Ticket	4072
SelectByIndex.....	4073
CPositionInfo	4074
Time	4076
TimeMsc	4077
TimeUpdate.....	4078
TimeUpdateMsc.....	4079
PositionType	4080
TypeDescription.....	4081
Magic	4082
Identifier	4083
Volume	4084
PriceOpen	4085
StopLoss	4086
TakeProfit	4087
PriceCurrent.....	4088
Commission.....	4089
Swap	4090
Profit	4091
Symbol	4092
Comment	4093
InfoInteger.....	4094
InfoDouble.....	4095
InfoString	4096
Select	4097
SelectByIndex.....	4098
SelectByMagic.....	4099
SelectByTicket.....	4100
StoreState.....	4101
CheckState.....	4102
CDealInfo	4103
Order	4105
Time	4106
TimeMsc	4107
DealType	4108
TypeDescription.....	4109
Entry	4110
EntryDescription.....	4111
Magic	4112
PositionId	4113
Volume	4114
Price	4115
Commision	4116
Swap	4117
Profit	4118

Symbol	4119
Comment	4120
InfoInteger.....	4121
InfoDouble.....	4122
InfoString	4123
Ticket	4124
SelectByIndex.....	4125
CTrade.....	4126
LogLevel	4130
SetExpertMagicNumber	4131
SetDeviationInPoints	4132
SetTypeFilling	4133
SetAsyncMode.....	4134
SetMarginMode	4135
OrderOpen	4136
OrderModify.....	4138
OrderDelete.....	4139
PositionOpen.....	4140
PositionModify	4141
PositionClose.....	4142
PositionClosePartial.....	4143
PositionCloseBy.....	4145
Buy	4146
Sell	4147
BuyLimit	4148
BuyStop	4149
SellLimit	4150
SellStop	4151
Request	4152
RequestAction	4153
RequestActionDescription.....	4154
RequestMagic	4155
RequestOrder.....	4156
RequestSymbol.....	4157
RequestVolume.....	4158
RequestPrice.....	4159
RequestStopLimit.....	4160
RequestSL	4161
RequestTP	4162
RequestDeviation	4163
RequestType	4164
RequestTypeDescription.....	4165
RequestTypeFilling	4166
RequestTypeFillingDescription.....	4167
RequestTypeTime.....	4168
RequestTypeTimeDescription.....	4169
RequestExpiration	4170
RequestComment.....	4171
RequestPosition	4172
RequestPositionBy	4173
Result	4174
ResultRetcode.....	4175
ResultRetcodeDescription.....	4176
ResultDeal	4177
ResultOrder	4178
ResultVolume.....	4179
ResultPrice	4180
ResultBid	4181

ResultAsk	4182
ResultComment	4183
CheckResult	4184
CheckResultRetcode	4185
CheckResultRetcodeDescription	4186
CheckResultBalance	4187
CheckResultEquity	4188
CheckResultProfit	4189
CheckResultMargin	4190
CheckResultMarginFree	4191
CheckResultMarginLevel	4192
CheckResultComment	4193
PrintRequest	4194
PrintResult	4195
FormatRequest	4196
FormatRequestResult	4197
CTerminalInfo	4198
Build	4200
IsConnected	4201
IsDLLsAllowed	4202
IsTradeAllowed	4203
IsEmailEnabled	4204
IsFtpEnabled	4205
MaxBars	4206
CodePage	4207
CPUCores	4208
MemoryPhysical	4209
MemoryTotal	4210
MemoryAvailable	4211
MemoryUsed	4212
IsX64	4213
OpenCLSupport	4214
DiskSpace	4215
Language	4216
Name	4217
Company	4218
Path	4219
DataPath	4220
CommonDataPath	4221
InfoInteger	4222
InfoString	4223
Strategy Modules	4224
Base classes for Expert Advisors	4227
CExpertBase	4228
InitPhase	4231
TrendType	4232
UsedSeries	4233
EveryTick	4234
Open	4235
High	4236
Low	4237
Close	4238
Spread	4239
Time	4240
TickVolume	4241
RealVolume	4242
Init	4243
Symbol	4244

Period	4245
Magic	4246
ValidationSettings	4247
SetPriceSeries	4248
SetOtherSeries	4249
InitIndicators	4250
InitOpen	4251
InitHigh	4252
InitLow	4253
InitClose	4254
InitSpread	4255
InitTime	4256
InitTickVolume	4257
InitRealVolume	4258
PriceLevelUnit	4259
StartIndex	4260
CompareMagic	4261
CExpert	4262
Init	4267
Magic	4268
InitSignal	4269
InitTrailing	4270
InitMoney	4271
InitTrade	4272
Deinit	4273
OnTickProcess	4274
OnTradeProcess	4275
OnTimerProcess	4276
OnChartEventProcess	4277
OnBookEventProcess	4278
MaxOrders	4279
Signal	4280
ValidationSettings	4281
InitIndicators	4282
OnTick	4283
OnTrade	4284
OnTimer	4285
OnChartEvent	4286
OnBookEvent	4287
InitParameters	4288
DeinitTrade	4289
DeinitSignal	4290
DeinitTrailing	4291
DeinitMoney	4292
DeinitIndicators	4293
Refresh	4294
Processing	4295
SelectPosition	4297
CheckOpen	4298
CheckOpenLong	4299
CheckOpenShort	4300
OpenLong	4301
OpenShort	4302
CheckReverse	4303
CheckReverseLong	4304
CheckReverseShort	4305
ReverseLong	4306
ReverseShort	4308

CheckClose.....	4310
CheckCloseLong.....	4311
CheckCloseShort.....	4312
CloseAll.....	4313
Close.....	4314
CloseLong.....	4315
CloseShort.....	4316
CheckTrailingStop.....	4317
CheckTrailingStopLong.....	4318
CheckTrailingStopShort.....	4319
TrailingStopLong.....	4320
TrailingStopShort.....	4321
CheckTrailingOrderLong.....	4322
CheckTrailingOrderShort.....	4323
TrailingOrderLong.....	4324
TrailingOrderShort.....	4325
CheckDeleteOrderLong.....	4326
CheckDeleteOrderShort.....	4327
DeleteOrders.....	4328
DeleteOrder.....	4329
DeleteOrderLong.....	4330
DeleteOrderShort.....	4331
LotOpenLong.....	4332
LotOpenShort.....	4333
LotReverse.....	4334
PrepareHistoryDate.....	4335
HistoryPoint.....	4336
CheckTradeState.....	4337
WaitEvent.....	4338
NoWaitEvent.....	4339
TradeEventPositionStopTake.....	4340
TradeEventOrderTriggered.....	4341
TradeEventPositionOpened.....	4342
TradeEventPositionVolumeChanged.....	4343
TradeEventPositionModified.....	4344
TradeEventPositionClosed.....	4345
TradeEventOrderPlaced.....	4346
TradeEventOrderModified.....	4347
TradeEventOrderDeleted.....	4348
TradeEventNotIdentified.....	4349
TimeframeAdd.....	4350
TimeframesFlags.....	4351
CExpertSignal.....	4352
BasePrice.....	4355
UsedSeries.....	4356
Weight.....	4357
PatternsUsage.....	4358
General.....	4359
Ignore.....	4360
Invert.....	4361
ThresholdOpen.....	4362
ThresholdClose.....	4363
PriceLevel.....	4364
StopLevel.....	4365
TakeLevel.....	4366
Expiration.....	4367
Magic.....	4368
ValidationSettings.....	4369

InitIndicators.....	4370
AddFilter	4371
CheckOpenLong.....	4372
CheckOpenShort	4373
OpenLongParams.....	4374
OpenShortParams.....	4375
CheckCloseLong.....	4376
CheckCloseShort	4377
CloseLongParams.....	4378
CloseShortParams.....	4379
CheckReverseLong	4380
CheckReverseShort.....	4381
CheckTrailingOrderLong.....	4382
CheckTrailingOrderShort.....	4383
LongCondition	4384
ShortCondition	4385
Direction	4386
CExpertTrailing.....	4387
CheckTrailingStopLong.....	4389
CheckTrailingStopShort.....	4390
CExpertMoney	4391
Percent	4392
ValidationSettings	4393
CheckOpenLong.....	4394
CheckOpenShort	4395
CheckReverse	4396
CheckClose.....	4397
Modules of Trade Signals	4398
Signals of the Indicator Accelerator Oscillator	4401
Signals of the Indicator Adaptive Moving Average.....	4404
Signals of the Indicator Awesome Oscillator.....	4408
Signals of the Oscillator Bears Power.....	4412
Signals of the Oscillator Bulls Power.....	4414
Signals of the Oscillator Commodity Channel Index	4416
Signals of the Oscillator DeMarker	4420
Signals of the Indicator Double Exponential Moving Average.....	4424
Signals of the Indicator Envelopes	4428
Signals of the Indicator Fractal Adaptive Moving Average	4431
Signals of the Intraday Time Filter	4435
Signals of the Oscillator MACD	4437
Signals of the Indicator Moving Average.....	4443
Signals of the Indicator Parabolic SAR.....	4447
Signals of the Oscillator Relative Strength Index.....	4449
Signals of the Oscillator Relative Vigor Index.....	4455
Signals of the Oscillator Stochastic	4457
Signals of the Oscillator Triple Exponential Average.....	4462
Signals of the Indicator Triple Exponential Moving Average.....	4466
Signals of the Oscillator Williams Percent Range.....	4470
Trailing Stop Classes.....	4473
CTrailingFixedPips.....	4474
StopLevel	4476
ProfitLevel.....	4477
ValidationSettings	4478
CheckTrailingStopLong.....	4479
CheckTrailingStopShort.....	4480
CTrailingMA.....	4481
Period	4483
Shift	4484

Method	4485
Applied	4486
InitIndicators.....	4487
ValidationSettings	4488
CheckTrailingStopLong.....	4489
CheckTrailingStopShort.....	4490
CTrailingNone	4491
CheckTrailingStopLong.....	4492
CheckTrailingStopShort.....	4493
CTrailingPSAR.....	4494
Step	4496
Maximum	4497
InitIndicators.....	4498
CheckTrailingStopLong.....	4499
CheckTrailingStopShort.....	4500
Money Management Classes.....	4501
CMoneyFixedLot.....	4502
Lots	4503
ValidationSettings	4504
CheckOpenLong.....	4505
CheckOpenShort	4506
CMoneyFixedMargin	4507
CheckOpenLong.....	4508
CheckOpenShort	4509
CMoneyFixedRisk.....	4510
CheckOpenLong.....	4511
CheckOpenShort	4512
CMoneyNone	4513
ValidationSettings	4514
CheckOpenLong.....	4515
CheckOpenShort	4516
CMoneySizeOptimized.....	4517
DecreaseFactor.....	4518
ValidationSettings	4519
CheckOpenLong.....	4520
CheckOpenShort	4521
Panels and Dialogs	4522
CRect	4524
Left	4525
Top	4526
Right	4527
Bottom	4528
Width	4529
Height	4530
SetBound	4531
Move	4532
Shift	4533
Contains	4534
Format	4535
CDateTime.....	4536
MonthName.....	4538
ShortMonthName.....	4539
DayName	4540
ShortDayName.....	4541
DaysInMonth.....	4542
DateTime	4543
Date	4544
Time	4545

Sec	4546
Min	4547
Hour	4548
Day	4549
Mon	4550
Year	4551
SecDec	4552
SecInc	4553
MinDec	4554
MinInc	4555
HourDec	4556
HourInc	4557
DayDec	4558
DayInc	4559
MonDec	4560
MonInc	4561
YearDec	4562
YearInc	4563
CWnd	4564
Create	4568
Destroy	4569
OnEvent	4570
OnMouseEvent	4571
Name	4572
ControlsTotal	4573
Control	4574
ControlFind	4575
Rect	4576
Left	4577
Top	4578
Right	4579
Bottom	4580
Width	4581
Height	4582
Move	4583
Shift	4584
Resize	4585
Contains	4586
Alignment	4587
Align	4588
Id	4589
IsEnabled	4590
Enable	4591
Disable	4592
IsVisible	4593
Visible	4594
Show	4595
Hide	4596
IsActive	4597
Activate	4598
Deactivate	4599
StateFlags	4600
StateFlagsSet	4601
StateFlagsReset	4602
PropFlags	4603
PropFlagsSet	4604
PropFlagsReset	4605
MouseX	4606

MouseY	4607
MouseFlags	4608
MouseFocusKill.....	4609
OnCreate	4610
OnDestroy.....	4611
OnMove	4612
OnResize	4613
OnEnable	4614
OnDisable	4615
OnShow	4616
OnHide	4617
OnActivate.....	4618
OnDeactivate.....	4619
OnClick	4620
OnChange	4621
OnMouseDown.....	4622
OnMouseUp.....	4623
OnDragStart	4624
OnDragProcess.....	4625
OnDragEnd.....	4626
DragObjectCreate.....	4627
DragObjectDestroy.....	4628
CWndObj	4629
OnEvent	4631
Text	4632
Color	4633
ColorBackground.....	4634
ColorBorder	4635
Font	4636
FontSize	4637
ZOrder	4638
OnObjectCreate	4639
OnObjectChange.....	4640
OnObjectDelete.....	4641
OnObjectDrag.....	4642
OnSetText	4643
OnSetColor	4644
OnSetColorBackground.....	4645
OnSetFont	4646
OnSetFontSize.....	4647
OnSetZOrder.....	4648
OnDestroy.....	4649
OnChange	4650
CWndContainer	4651
Destroy	4653
OnEvent	4654
OnMouseEvent.....	4655
ControlsTotal.....	4656
Control	4657
ControlFind.....	4658
Add	4659
Delete	4660
Move	4661
Shift	4662
Id	4663
Enable	4664
Disable	4665
Show	4666

Hide	4667
MouseFocusKill.....	4668
Save	4669
Load	4670
OnResize	4671
OnActivate.....	4672
OnDeactivate.....	4673
CLabel	4674
Create	4679
OnSetText.....	4680
OnSetColor	4681
OnSetFont	4682
OnSetFontSize.....	4683
OnCreate	4684
OnShow	4685
OnHide	4686
OnMove	4687
CBmpButton.....	4688
Create	4694
Border	4695
BmpNames	4696
BmpOffName.....	4697
BmpOnName.....	4698
BmpPassiveName	4699
BmpActiveName.....	4700
Pressed	4701
Locking	4702
OnSetZOrder.....	4703
OnCreate	4704
OnShow	4705
OnHide	4706
OnMove	4707
OnChange	4708
OnActivate.....	4709
OnDeactivate.....	4710
OnMouseDown	4711
OnMouseUp.....	4712
CButton.....	4713
Create	4720
Pressed	4721
Locking	4722
OnSetText.....	4723
OnSetColor	4724
OnSetColorBackground.....	4725
OnSetColorBorder.....	4726
OnSetFont	4727
OnSetFontSize.....	4728
OnCreate	4729
OnShow	4730
OnHide	4731
OnMove	4732
OnResize	4733
OnMouseDown	4734
OnMouseUp.....	4735
CEdit	4736
Create	4741
ReadOnly	4742
TextAlign	4743

OnObjectEndEdit.....	4744
OnSetText.....	4745
OnSetColor.....	4746
OnSetColorBackground.....	4747
OnSetColorBorder.....	4748
OnSetFont.....	4749
OnSetFontSize.....	4750
OnSetZOrder.....	4751
OnCreate.....	4752
OnShow.....	4753
OnHide.....	4754
OnMove.....	4755
OnResize.....	4756
OnChange.....	4757
OnClick.....	4758
CPanel.....	4759
Create.....	4764
BorderType.....	4765
OnSetText.....	4766
OnSetColorBackground.....	4767
OnSetColorBorder.....	4768
OnCreate.....	4769
OnShow.....	4770
OnHide.....	4771
OnMove.....	4772
OnResize.....	4773
OnChange.....	4774
CPicture.....	4775
Create.....	4780
Border.....	4781
BmpName.....	4782
OnCreate.....	4783
OnShow.....	4784
OnHide.....	4785
OnMove.....	4786
OnChange.....	4787
CScroll.....	4788
Create.....	4790
OnEvent.....	4791
MinPos.....	4792
MaxPos.....	4793
CurrPos.....	4794
CreateBack.....	4795
CreateInc.....	4796
CreateDec.....	4797
CreateThumb.....	4798
OnClickInc.....	4799
OnClickDec.....	4800
OnShow.....	4801
OnHide.....	4802
OnChangePos.....	4803
OnThumbDragStart.....	4804
OnThumbDragProcess.....	4805
OnThumbDragEnd.....	4806
CalcPos.....	4807
CScrollV.....	4808
CreateInc.....	4814
CreateDec.....	4815

CreateThumb.....	4816
OnResize	4817
OnChangePos	4818
OnThumbDragStart	4819
OnThumbDragProcess	4820
OnThumbDragEnd	4821
CalcPos	4822
CScrollH.....	4823
CreateInc	4829
CreateDec.....	4830
CreateThumb.....	4831
OnResize	4832
OnChangePos	4833
OnThumbDragStart	4834
OnThumbDragProcess	4835
OnThumbDragEnd	4836
CalcPos	4837
CWndClient.....	4838
Create	4841
OnEvent	4842
ColorBackground.....	4843
ColorBorder	4844
BorderStyle.....	4845
VScrolled	4846
HScrolled	4847
CreateBack.....	4848
CreateScrollV	4849
CreateScrollH.....	4850
OnResize	4851
OnVScrollShow.....	4852
OnVScrollHide.....	4853
OnHScrollShow.....	4854
OnHScrollHide.....	4855
OnScrollLineDown	4856
OnScrollLineUp.....	4857
OnScrollLineLeft.....	4858
OnScrollLineRight.....	4859
Rebound	4860
CListView.....	4861
Create	4867
OnEvent	4868
TotalView	4869
AddItem	4870
Select	4871
SelectByText	4872
SelectByValue.....	4873
Value	4874
CreateRow.....	4875
OnResize	4876
OnVScrollShow.....	4877
OnVScrollHide.....	4878
OnScrollLineDown	4879
OnScrollLineUp.....	4880
OnItemClick.....	4881
Redraw	4882
RowState	4883
CheckView.....	4884
CComboBox.....	4885

Create	4891
OnEvent	4892
AddItem	4893
ListViewItems	4894
Select	4895
SelectByText	4896
SelectByValue	4897
Value	4898
CreateEdit	4899
CreateButton	4900
CreateList	4901
OnClickEdit	4902
OnClickButton	4903
OnChangeList	4904
ListShow	4905
ListHide	4906
CCheckBox	4907
Create	4913
OnEvent	4914
Text	4915
Color	4916
Checked	4917
Value	4918
CreateButton	4919
CreateLabel	4920
OnClickButton	4921
OnClickLabel	4922
CCheckGroup	4923
Create	4929
OnEvent	4930
AddItem	4931
Value	4932
CreateButton	4933
OnVScrollShow	4934
OnVScrollHide	4935
OnScrollLineDown	4936
OnScrollLineUp	4937
OnChangeItem	4938
Redraw	4939
RowState	4940
CRadioButton	4941
Create	4943
OnEvent	4944
Text	4945
Color	4946
State	4947
CreateButton	4948
CreateLabel	4949
OnClickButton	4950
OnClickLabel	4951
CRadioGroup	4952
Create	4958
OnEvent	4959
AddItem	4960
Value	4961
CreateButton	4962
OnVScrollShow	4963
OnVScrollHide	4964

OnScrollLineDown	4965
OnScrollLineUp	4966
OnChangeltem	4967
Redraw	4968
RowState	4969
Select	4970
CSpinEdit	4971
Create	4976
OnEvent	4977
MinValue	4978
MaxValue	4979
Value	4980
CreateEdit	4981
CreateInc	4982
CreateDec	4983
OnClickInc	4984
OnClickDec	4985
OnChangeValue	4986
CDialog	4987
Create	4989
OnEvent	4990
Caption	4991
Add	4992
CreateWhiteBorder	4993
CreateBackground	4994
CreateCaption	4995
CreateButtonClose	4996
CreateClientArea	4997
OnClickCaption	4998
OnClickButtonClose	4999
ClientAreaVisible	5000
ClientAreaLeft	5001
ClientAreaTop	5002
ClientAreaRight	5003
ClientAreaBottom	5004
ClientAreaWidth	5005
ClientAreaHeight	5006
OnDialogDragStart	5007
OnDialogDragProcess	5008
OnDialogDragEnd	5009
CAppDialog	5010
Create	5013
Destroy	5014
OnEvent	5015
Run	5016
ChartEvent	5017
Minimized	5018
IniFileSave	5019
IniFileLoad	5020
IniFileName	5021
IniFileExt	5022
CreateCommon	5023
CreateExpert	5024
CreateIndicator	5025
CreateButtonMinMax	5026
OnClickButtonClose	5027
OnClickButtonMinMax	5028
OnAnotherApplicationClose	5029

	Rebound	5030
	Minimize	5031
	Maximize	5032
	CreateInstanceId.....	5033
	ProgramName.....	5034
	SubwinOff	5035
27	Moving from MQL4.....	5036
28	List of MQL5 Functions	5039
29	List of MQL5 Constants.....	5064

MQL5 Reference

MetaQuotes Language 5 (MQL5) is a high-level language designed for developing technical indicators, trading robots and utility applications, which automate financial trading. MQL5 has been developed by [MetaQuotes Software Corp.](#) for their trading platform. The language syntax is very close to C++ enabling programmers to develop applications in the object-oriented programming (OOP) style.

In addition to the MQL5 language, the trading platform package also includes the [MetaEditor IDE](#) with highly advanced code writing tools, such as templates, snippets, debugging, profiling and auto completion tools, as well as built-in [MQL5 Storage](#) enabling file versioning.

The language support is available on the MQL5.community website, which contains a huge [free CodeBase](#) and a plethora of [articles](#). These articles cover all the aspects of the modern trading, including neural networks, statistics and analysis, high-frequency trading, arbitrage, testing and optimization of trading strategies, use of trading automation robots, and more.

Traders and MQL5 program developers can communicate on the forum, order and develop applications using the [Freelance](#) service, as well as buy and sell protected programs in the [Market](#) of automated trading applications.

The MQL5 language provides specialized [trading functions](#) and predefined [event handlers](#) to help programmers develop Expert Advisors (EAs), which automatically control trading processes following specific trading rules. In addition to EAs, MQL5 allows developing custom [technical indicators](#), scripts and libraries.

This MQL5 language reference contains functions, operations, reserved words and other language constructions divided into categories. The reference also provides descriptions of [Standard Library](#) classes used for developing trading strategies, control panels, custom graphics and enabling file access.

Additionally, the CodeBase contains the [ALGLIB](#) numerical analysis library, which can be used for solving various mathematical problems.

Types of MQL5 Applications

MQL5 programs are divided into four specialized types based on the trading automation tasks that they implement:

- **Expert Advisor** is an automated trading system linked to a chart. An Expert Advisor contains [event](#) handlers to manage predefined events which activate execution of appropriate trading strategy elements. For example, an event of program initialization and deinitialization, new ticks, timer events, changes in the Depth of Market, chart and custom events.
In addition to calculating trading signals based on the implemented rules, Expert Advisors can also automatically execute trades and send them directly to a trading server. Expert Advisors are stored in `<Terminal_Directory>\MQL5\Experts`.
- **Custom Indicators** is a technical indicator developed by a user in addition to standard indicators integrated into the trading platform. Custom indicators, as well as standard ones, cannot trade automatically, but only implement analytical functions. Custom indicators can utilize values of other indicators for calculations, and can be called from Expert Advisors.
Custom indicators are stored in `<Terminal_Directory>\MQL5\Indicators`.

- **Script** is a program for a single execution of an action. Unlike Expert Advisors, scripts do not handle any event except for trigger, initialization and deinitialization. A script code must contain the OnStart handler function.
Scripts are stored in `<Terminal_Directory>\MQL5\Scripts`.
- **Library** is a set of custom functions. Libraries are intended to store and distribute commonly used algorithms of custom programs.
Libraries are stored in `<Terminal_Directory>\MQL5\Libraries`.
- **Include File** is a source text of the most frequently used blocks of custom programs. Such files can be included into the source texts of Expert Advisors, scripts, custom indicators, and libraries at the compiling stage. The use of included files is more preferable than the use of libraries because of additional burden occurring at calling library functions.
Include files can be stored in the same directory where the original file is located. In this case the `#include` directive with double quotes is used. Another option is to store include files in `<Terminal_Directory>\MQL5\Include`. In this case `#include` with angle brackets should be used.

© 2000-2017, [MetaQuotes Software Corp.](#)

Language Basics

The MetaQuotes Language 5 (MQL5) is an object-oriented high-level programming language intended for writing automated trading strategies, custom technical indicators for the analysis of various financial markets. It allows not only to write a variety of expert systems, designed to operate in real time, but also create their own graphical tools to help you make trade decisions.

MQL5 is based on the concept of the popular programming language C++. As compared to MQL4, the new language now has [enumerations](#), [structures](#), [classes](#) and [event handling](#). By increasing the number of embedded main [types](#), the interaction of executable programs in MQL5 with other applications through dll is now as easy as possible. MQL5 syntax is similar to the syntax of C++, and this makes it easy to translate into it programs from modern programming languages.

To help you study the MQL5 language, all topics are grouped into the following sections:

- [Syntax](#)
- [Data Types](#)
- [Operations and Expressions](#)
- [Operators](#)
- [Functions](#)
- [Variables](#)
- [Preprocessor](#)
- [Object-Oriented Programming](#)

Syntax

As to the syntax, THE MQL5 language for programming trading strategies is very much similar to the C++ programming language, except for some features:

- no address arithmetic;
- no goto operator;
- an anonymous enumeration can't be declared;
- no multiple inheritance.

See also

[Enumerations](#), [Structures and Classes](#), [Inheritance](#)

Comments

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

Examples:

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


Identifiers

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

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

The identifier must not coincide with [reserved](#) word.

Examples:

```
NAME1 name1 Total_5 Paper
```

See also

[Variables](#), [Functions](#)

Reserved Words

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

Data Types

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

Access Specificators

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

Memory Classes

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

Operators

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

Other

<u>false</u>	<u>#define</u>	<u>#property</u>
<u>this</u>	<u>#import</u>	<u>template</u>

true#includetypename

Data Types

Any program operates with data. Data can be of different types depending on their purposes. For example, integer data are used to access to array components. Price data belong to those of double precision with floating point. This is related to the fact that no special data type for price data is provided in MQL5.

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

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

The basic data types are:

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

Complex data types are:

- [structures](#);
- [classes](#).

In terms of [OOP](#) complex data types are called abstract data types.

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

Only implicit [type casting](#) is used in [expressions](#), unless the explicit casting is specified.

See also

[Typecasting](#)

Integer Types

In MQL5 integers are represented by eleven types. Some types can be used together with other ones, if required by the program logic, but in this case it's necessary to remember the rules of [typecasting](#).

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

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

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

See also

[Conversion Functions](#), [Numerical Type Constants](#)

Char, Short, Int and Long Types

char

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

uchar

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

short

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

ushort

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

int

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

uint

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

long

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

ulong

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

Examples:

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


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

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

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

The correct variant is:

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

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

Result:

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


...

Examples:

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

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

Examples:

```
0x0A, 0x12, 0X12, 0x2f, 0xA3, 0xA3, 0x7C7
```

For integer variables, the values can be set in binary form using B prefix. For example, you can encode the working hours of a trading session into `int` type variable and use information about them according to the required algorithm:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- set 1 for working hours and 0 for nonworking ones
    int AsianSession   =B'11111111'; // Asian session from 0:00 to 9:00
    int EuropeanSession=B'111111110000000000'; // European session 9:00 - 18:00
    int AmericanSession =B'111111110000000000000011'; // American session 16:00 - 02:00
//--- derive numerical values of the sessions
    PrintFormat("Asian session hours as value =%d",AsianSession);
    PrintFormat("European session hours as value is %d",EuropeanSession);
    PrintFormat("American session hours as value is %d",AmericanSession);
//--- and now let's display string representations of the sessions' working hours
    Print("Asian session ",GetHoursForSession(AsianSession));
    Print("European session ",GetHoursForSession(EuropeanSession));
    Print("American session ",GetHoursForSession(AmericanSession));
//---
}
//+-----+
//| return the session's working hours as a string |
//+-----+
string GetHoursForSession(int session)
{
//--- in order to check, use AND bit operations and left shift by 1 bit <=>=1
//--- start checking from the lowest bit
    int bit=1;
    string out="working hours: ";
//--- check all 24 bits starting from the zero and up to 23 inclusively
    for(int i=0;i<24;i++)
    {
        //--- receive bit state in number
```



```
bool workinghour=(session&bit)==bit;
//--- add the hour's number to the message
if(workinghour )out=out+StringFormat("%d ",i);
//--- shift by one bit to the left to check the value of the next one
bit<<=1;
}
//--- result string
return out;
}
```

See also

[Typecasting](#)

Character Constants

Characters as elements of a [string](#) in MQL5 are indexes in the Unicode character set. They are hexadecimal values that can be cast into integers, and that can be manipulated by integer [operations](#) like addition and subtraction.

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

Example:

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

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

Character name	Mnemonic code or image	Record in MQL5	Numeric value
new line (line feed)	LF	'\n'	10
horizontal tab	HT	'\t'	9
carriage return	CR	'\r'	13
backslash	\	'\\'	92
single quote	'	'\"'	39
double quote	"	'\"'	34
hexadecimal code	hhhh	'\xhhhh'	1 to 4 hexadecimal characters
decimal code	d	'\d'	decimal number from 0 to 65535

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

Example

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

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

```
void OnStart()
{
```



```

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

```

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

See also

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

Datetime Type

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

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

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

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

Examples:

```
datetime NY=D'2015.01.01 00:00';      // Time of beginning of year 2015
datetime d1=D'1980.07.19 12:30:27';   // Year Month Day Hours Minutes Seconds
datetime d2=D'19.07.1980 12:30:27';   // Equal to D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';         // Equal to D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';           // Equal to D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;    // Compilation date
datetime compilation_date_time=__DATETIME__; // Compilation date and time
datetime compilation_time=__DATETIME__-__DATE__; // Compilation time
//--- Examples of declarations after which compiler warnings will be returned
datetime warning1=D'12:30:27';        // Equal to D'[date of compilation] 12:30:27'
datetime warning2=D'';                // Equal to __DATETIME__
```

See also

[Structure of the Date Type](#), [Date and Time](#), [TimeToString](#), [StringToTime](#)

Color Type

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

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

Literal representation consists of three parts representing numerical rate values of the three main color components: red, green, blue. The constant starts with `C` and is enclosed in single quotes. Numerical rate values of a color component lie in the range from 0 to 255.

Integer-valued representation is written in a form of hexadecimal or a decimal number. A hexadecimal number looks like `0x00BBGGRR`, where `RR` is the rate of the red color component, `GG` - of the green one, and `BB` - of the blue one. Decimal constants are not directly reflected in the RGB. They represent a decimal value of the hexadecimal integer representation.

Specific colors reflect the so-called [Web-colors](#) set.

Examples:

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

See also

[Web Colors](#), [ColorToString](#), [StringToColor](#), [Typecasting](#)

Bool Type

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

Examples:

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

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

Examples:

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

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

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

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

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

See also

[Boolean Operations](#), [Precedence Rules](#)

Enumerations

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

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

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

Example:

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

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

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

Example:

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

Notes

- Unlike C++, the size of the internal representation of the enumerated type in MQL5 is always equal to 4 bytes. That is, [sizeof](#) (months) returns the value 4.
- Unlike C++, an anonymous enumeration can't be declared in MQL5. That is, a unique name must be always specified after the enum keyword.

See also

[Typecasting](#)

Real Types (double, float)

Real types (or floating-point types) represent values with a fractional part. In the MQL5 language there are two types for floating point numbers. The method of representation of real numbers in the computer memory is defined by the IEEE 754 standard and is independent of platforms, operating systems or programming languages.

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

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

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

Examples:

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

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

Example:

[illegible]

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

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

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

Example:

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

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

Example:

```
bool EqualDoubles(double d1,double d2,double epsilon)
{
    if(epsilon<0) epsilon=-epsilon;
    //---
    if(d1-d2>epsilon) return false;
    if(d1-d2<-epsilon) return false;
    //---
    return true;
}

void OnStart()
{
    double d_val=0.7;
    float f_val=0.7;
    if(EqualDoubles(d_val,f_val,0.0000000000000001)) Print(d_val," equals ",f_val);
    else Print("Different: d_val = ",DoubleToString(d_val,16),
              " f_val = ",DoubleToString(f_val,16));
    // Result: Different: d_val= 0.7000000000000000    f_val= 0.6999999880790710
}
```

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

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

Example:

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

void OnStart()
{
    double d_val = 0.3;
    float f_val = 0.3;
    if(CompareDoubles(d_val, f_val)) Print(d_val, " equals ", f_val);
    else Print("Different: d_val = ", DoubleToString(d_val, 16),
              " f_val = ", DoubleToString(f_val, 16));
    // Result: Different: d_val= 0.3000000000000000    f_val= 0.3000000119209290
}
```

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

Example:

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

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

Examples:

```
struct str1
{
    double d;
};

struct str2
{
    long l;
};

//--- Start
str1 s1;
str2 s2;
```



```

//---
s1.d=MathArcsin(2.0);          // Get the invalid number -1.#IND
s2=s1;
printf("1.  %f %I64X",s1.d,s2.l);
//---
s2.l=0xFFFFF0000000000000;    // invalid number -1.#QNAN
s1=s2;
printf("2.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FF7000000000000;      // greatest non-number SNaN
s1=s2;
printf("3.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FF8000000000000;      // smallest non-number QNaN
s1=s2;
printf("4.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FFF000000000000;      // greatest non-number QNaN
s1=s2;
printf("5.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FF0000000000000;      // Positive infinity 1.#INF and smallest non-number SNaN
s1=s2;
printf("6.  %f %I64X",s1.d,s2.l);
//---
s2.l=0xFFFF000000000000;      // Negative infinity -1.#INF
s1=s2;
printf("7.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x8000000000000000;      // Negative zero -0.0
s1=s2;
printf("8.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x3FE0000000000000;      // 0.5
s1=s2;
printf("9.  %f %I64X",s1.d,s2.l);
//---
s2.l=0x3FF0000000000000;      // 1.0
s1=s2;
printf("10. %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FEFFFFFFFFFFFFFFF;    // Greatest normalized number (MAX_DBL)
s1=s2;
printf("11. %.16e %I64X",s1.d,s2.l);
//---
s2.l=0x0010000000000000;      // Smallest positive normalized (MIN_DBL)
s1=s2;
printf("12. %.16e %.16I64X",s1.d,s2.l);
//---

```



```
s1.d=0.7; // Show that the number of 0.7 - endless fraction
s2=s1;
printf("13. %.16e %.16I64X",s1.d,s2.l);
/*
1. -1.#IND00 FFF8000000000000
2. -1.#QNAN0 FFFF000000000000
3. 1.#SNAN0 7FF7000000000000
4. 1.#QNAN0 7FF8000000000000
5. 1.#QNAN0 7FFF000000000000
6. 1.#INF00 7FF0000000000000
7. -1.#INF00 FFF0000000000000
8. -0.000000 8000000000000000
9. 0.500000 3FE0000000000000
10. 1.000000 3FF0000000000000
11. 1.7976931348623157e+308 7FEFFFFFFFFFFFFFFF
12. 2.2250738585072014e-308 0010000000000000
13. 6.9999999999999996e-001 3FE6666666666666
*/
```

See also

[DoubleToString](#), [NormalizeDouble](#), [Numeric Type Constants](#)

String Type

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

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

Examples:

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

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

```
//--- Declare a long constant string
string HTML_head="<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN\"
    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n"
    "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
    "<head>\n"
    "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">\n"
    "<title>Trade Operations Report</title>\n"
    "</head>";

//--- Output the constant string into log
Print(HTML_head);
}
```

See also

[Conversion Functions](#), [String Functions](#), [FileOpen](#), [FileReadString](#), [FileWriteString](#)

Structures, Classes and Interfaces

Structures

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

Structure Declaration

The structure data type is determined by the following description:

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

The structure name can't be used as an identifier (name of a variable or function). It should be noted that in MQL5 structure elements follow one another directly, without alignment. In C++ such an order is made to the compiler using the following instruction:

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

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

Example:

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

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

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

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

Simple Structures

Structures that do not contain strings or objects of dynamic arrays are called simple structures; variables of such structures can be [freely copied](#) to each other, even if they are different structures.

Variables of simple structures, as well as their array can be passed as parameters to functions [imported](#) from DLL.

Access to Structure Members

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

Example:

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

Modifier 'final'

The use of the 'final' modifier during structure declaration prohibits further inheritance from this structure. If a structure requires no further modifications, or modifications are not allowed for security reasons, declare this structure with the 'final' modifier. In addition, all the members of the structure will also be implicitly considered final.

```
struct settings final
{
    //--- Structure body
};

struct trade_settings : public settings
{
    //--- Structure body
};
```

If you try to inherit from a structure with the 'final' modifier as shown in the above example, the compiler will return an error:

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

Classes

Classes differ from structures in the following:

- the keyword class is used in declaration;
- by default, all class members have access specifier private, unless otherwise indicated. Data-members of the structure have the default type of access as public, unless otherwise indicated;
- class objects always have a table of [virtual functions](#), even if there are no virtual functions declared in the class. Structures can't have virtual functions;

- the [new](#) operator can be applied to class objects; this operator cannot be applied to structures;
- classes can be [inherited](#) only from classes, structures can be inherited only from structures.

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

Example:

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

Constructors and Destructors

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

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

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

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

```
//+-----+
//| A class for working with a date |
//+-----+
class MyDateClass
{
private:
    int      m_year;           // Year
    int      m_month;          // Month
    int      m_day;            // Day of the month
    int      m_hour;           // Hour in a day
    int      m_minute;         // Minutes
    int      m_second;         // Seconds
public:
    //-- Default constructor
```



```

        MyDateClass(void);

    //--- Parametric constructor
        MyDateClass(int h,int m,int s);

};

```

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

```

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

```

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

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

```

//+-----+

```



```

//| A class with a default constructor |
//+-----+
class CFoo
{
    datetime      m_call_time;      // Time of the last object call
public:
    ///--- Constructor with a parameter that has a default value is not a default constructor
    CFoo(const datetime t=0){m_call_time=t;};

    ///--- Copy constructor
    CFoo(const CFoo &foo){m_call_time=foo.m_call_time;};

    string ToString(){return(TimeToString(m_call_time,TIME_DATE|TIME_SECONDS));};
};
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    // CFoo foo; // This variant cannot be used - a default constructor is not set
    ///--- Possible options to create the CFoo object
    CFoo foo1(TimeCurrent());        // An explicit call of a parametric constructor
    CFoo foo2();                     // An explicit call of a parametric constructor with
    CFoo foo3=D'2009.09.09';         // An implicit call of a parametric constructor
    CFoo foo4(foo1);                  // An explicit call of a copy constructor
    CFoo foo41=foo1;                  // An implicit call of a copy constructor
    CFoo foo5;                        // An explicit call of a default constructor (if there is no
                                    // then a parametric constructor with a default value
    ///--- Possible options to receive CFoo pointers
    CFoo *pfoo6=new CFoo();           // Dynamic creation of an object and receiving of a pointer
    CFoo *pfoo7=new CFoo(TimeCurrent()); // Another option of dynamic object creation
    CFoo *pfoo8=GetPointer(foo1);     // Now pfoo8 points to object foo1
    CFoo *pfoo9=pfoo7;                // pfoo9 and pfoo7 point to one and the same object
    // CFoo foo_array[3];              // This option cannot be used - a default constructor is not set
    ///--- Show the value of m_call_time
    Print("foo1.m_call_time=",foo1.ToString());
    Print("foo2.m_call_time=",foo2.ToString());
    Print("foo3.m_call_time=",foo3.ToString());
    Print("foo4.m_call_time=",foo4.ToString());
    Print("foo5.m_call_time=",foo5.ToString());
    Print("pfoo6.m_call_time=",pfoo6.ToString());
    Print("pfoo7.m_call_time=",pfoo7.ToString());
    Print("pfoo8.m_call_time=",pfoo8.ToString());
    Print("pfoo9.m_call_time=",pfoo9.ToString());
    ///--- Delete dynamically created arrays
    delete pfoo6;
    delete pfoo7;
    //delete pfoo8; // You do not need to delete pfoo8 explicitly, since it points to
    //delete pfoo9; // You do not need to delete pfoo9 explicitly. since it points to
}

```


If you uncomment these strings

```
//Cfoo foo_array[3];      // This variant cannot be used - a default constructor is not defined
```

or

```
//Cfoo foo_dyn_array[];  // This variant cannot be used - a default constructor is not defined
```

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

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

```
//+-----+
//| A class without a default constructor |
//+-----+
class Cfoo
{
    string      m_name;
public:
    Cfoo(string name) { m_name=name; }

};
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //-- Get the "default constructor is not defined" error during compilation
    Cfoo badFoo[5];
}
```

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

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

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

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

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

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

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


In this case, the CPerson class has three constructors:

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

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

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

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

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

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


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

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

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

Defining Class Methods

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

Example:

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

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

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

Example:

```
//+-----+
//| Constructor of the basic class |
//+-----+
void CTetrisShape::CTetrisShape()
{
    m_type=0;
    m_ypos=0;
    m_xpos=0;
    m_xsize=SHAPE_SIZE;
    m_ysize=SHAPE_SIZE;
    m_prev_turn=0;
    m_turn=0;
    m_right_border=0;
}

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

Public, Protected and Private Access Modifiers

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

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

Example:

```
class CTetrisField
{
private:
```



```

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

```

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

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

Access to the members of the basis class can be redefined during [inheritance](#) in derived classes.

Modifier 'final'

The use of the 'final' modifier during class declaration prohibits further inheritance from this class. If the class interface requires no further modifications, or modifications are not allowed for security reasons, declare this class with the 'final' modifier. In addition, all the members of the class will also be implicitly considered final.

```

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

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

```

If you try to inherit from a class with the 'final' modifier as shown in the above example, the compiler will return an error:

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

Interfaces

An interface allows determining specific functionality, which a class can then implement. In fact, an interface is a class that cannot contain any members, and may not have a constructor and/or a destructor. All methods declared in an interface are purely virtual, even without an explicit definition.

An interface is defined using the "interface" keyword. Example:

```
//--- Basic interface for describing animals
interface IAnimal
{
//--- The methods of the interface have public access by default
    void Sound(); // The sound produced by the animal
};
//+-----+
//| The CCat class is inherited from the IAnimal interface |
//+-----+
class CCat : public IAnimal
{
public:
    CCat() { Print("Cat was born"); }
    ~CCat() { Print("Cat is dead"); }

    //--- Implementing the Sound method of the IAnimal interface
    void Sound() { Print("meou"); }
};
//+-----+
//| The CDog class is inherited from the IAnimal interface |
//+-----+
class CDog : public IAnimal
{
public:
    CDog() { Print("Dog was born"); }
    ~CDog() { Print("Dog is dead"); }

    //--- Implementing the Sound method of the IAnimal interface
    void Sound() { Print("guaf"); }
};
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- An array of pointers to objects of the IAnimal type
    IAnimal *animals[2];
//--- Creating child classes of IAnimal and saving pointers to them into an array
    animals[0]=new CCat;
    animals[1]=new CDog;
//--- Calling the Sound() method of the basic IAnimal interface for each child
```



```
    for(int i=0;i<ArraySize(animals);++i)
        animals[i].Sound();
//--- Deleting objects
    for(int i=0;i<ArraySize(animals);++i)
        delete animals[i];
//--- Execution result
/*
    Cat was born
    Dog was born
    meou
    guaf
    Cat is dead
    Dog is dead
*/
}
```

Like with [abstract classes](#), an interface object cannot be created without inheritance. An interface can only be inherited from other interfaces and can be a parent for a class. An interface is always [publicly visible](#).

An interface cannot be declared within a class or structure declaration, but a pointer to the interface can be saved in a variable of type [void *](#). Generally speaking, a pointer to an object of any class can be saved into a variable of type [void *](#). In order to convert a void * pointer to a pointer to an object of a particular class, use the [dynamic_cast](#) operator. If conversion is not possible, the result of the dynamic_cast operation will be [NULL](#).

See also

[Object-Oriented Programming](#)

Dynamic Array Object

Dynamic Arrays

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

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

Example:

```
double matrix[][10][20]; // 3-dimensional dynamic array
ArrayResize(matrix,5);    // Set the size of the first dimension
```

Static Arrays

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

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

Examples:

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

Arrays in Structures

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

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

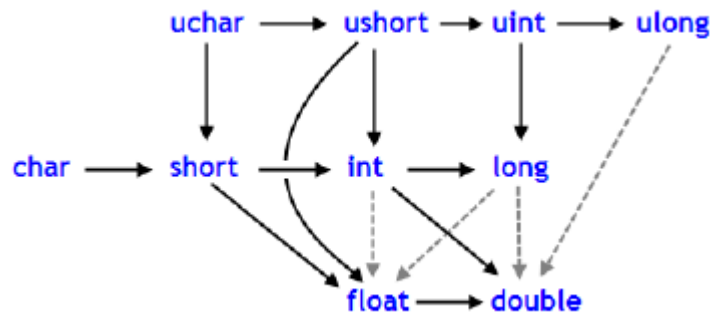
See also

[Array Functions](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Typecasting

Casting Numeric Types

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



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

```

int n=123456789;
float f=n;      // the content of f is equal to 1.234567892E8
Print("n = ",n,"    f = ",f);
// result n= 123456789    f= 123456792.00000

```

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

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

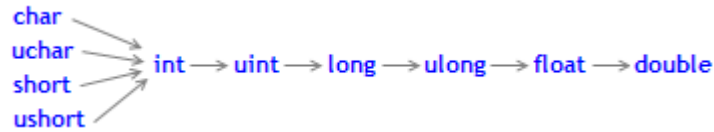
Example:

```

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

```


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



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

Examples:

```

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

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

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

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

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

Typcasting of Numeric Types

In the expressions of the MQL5 language both explicit and implicit typcasting can be used. The explicit typcasting is written as follows:

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

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

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

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

```

//--- Third example
double d2=(double)c1/2+0.3;
  
```



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

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

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

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

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

Examples:

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

int k=1/2.0; // the expression is cast to the double type,
Print("k = 1/2 ",k); // then is to the target type of int, the result is 0

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

double e=1/2.0; // the expression is cast to the double type,
Print("e = 1/2.0; ",e); // that is the same as the target type, the result is 0.5

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

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

```
void OnStart()
{
    long l_max=LONG_MAX;
    long l_min=LONG_MIN+1;
    //--- define the highest integer value, which does not lose accuracy when being cast to double
    while(l_max!=long((double)l_max))
        l_max--;
    //--- define the lowest integer value, which does not lose accuracy when being cast to double
    while(l_min!=long((double)l_min))
        l_min++;
    //--- derive the found interval for integer values
```



```

PrintFormat("When casting an integer value to double, it must be "
            "within [%I64d, %I64d] interval", l_min, l_max);
//--- now, let's see what happens if the value falls out of this interval
PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))=%I64d",
            l_max+1, double(l_max+1), long(double(l_max+1)));
PrintFormat("l_min-1=%I64d, double(l_min-1)=%.f, ulong(double(l_min-1))=%I64d",
            l_min-1, double(l_min-1), long(double(l_min-1)));
//--- receive the following result
// When casting an integer value to double, it should be within [-9223372036854774784,
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong(double(l_max+1))=9223372036854774785,
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulong(double(l_min-1))=9223372036854774785
}

```

Typecasting for the String Type

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

Examples:

```

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

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

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

```

Typecasting of Simple Structure Types

Data of the [simple structures](#) type can be assigned to each other only if all the members of both structures are of numeric types. In this case both operands of the [assignment operation](#) (left and right) must be of the structures type. The member-wise casting is not performed, a simple copying is

done. If the structures are of different sizes, the number of bytes of the smaller size is copied. Thus the absence of union in MQL5 is compensated.

Examples:

```

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

```

Another example illustrates the method of organizing a custom function for receiving RGB (Red, Green, Blue) representation from the [color](#) type. Create two structures of the same size but with

different contents. For convenience, let's add a function returning the RGB representation of a color as a string.

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

Typecasting of Base Class Pointers to Pointers of Derivative Classes

Objects of the [open generated](#) class can also be viewed as objects of the corresponding base class. This leads to some interesting consequences. For example, despite the fact that objects of different classes, generated by a single base class, may differ significantly from each other, we can create a

linked list (List) of them, as we view them as objects of the base type. But the converse is not true: the base class objects are not automatically objects of a derived class.

You can use the explicit casting to convert the base class pointers to the [pointers](#) of a derived class. But you must be fully confident in the admissibility of such a transformation, because otherwise a critical runtime error will occur and the mql5 program will be stopped.

Dynamic typecasting using `dynamic_cast` operator

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

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

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

Example:

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

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

See also

[Data Types](#)

Void Type and NULL Constant

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

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

Example:

```
//--- If the string is not initialized, then assign our predefined value to it
if(some_string==NULL) some_string="empty";
```

Also **NULL** can be compared to pointers to objects created with the [new operator](#).

See also

[Variables](#), [Functions](#)

User-defined types

The `typedef` keyword in C++ allows creating user-defined data types. To do this, simply specify a new data type name for an already existing data type. The new data type is not created. A new name for the existing type is defined instead. User-defined types make applications more flexible: sometimes, it is enough to change typedef instructions using substitution macros (`#define`). User-defined types also improve code readability since it is possible to apply custom names to standard data types using `typedef`. The general format of the entry for creating a user-defined type:

```
typedef type new_name;
```

Here, *type* means any acceptable data type, while *new_name* is a new name of the type. A new name is set only as an addition (not as a replacement) to an existing type name. MQL5 allows creating pointers to functions using `typedef`.

Pointer to the function

A pointer to a function is generally defined in the following format

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

where after `typedef`, the function signature (number and type of input parameters, as well as a type of a result returned by the function) is set. Below is a simple example of creating and applying a pointer to a function:

```
//--- declare a pointer to a function that accepts two int parameters
typedef int (*TFunc)(int,int);
//--- TFunc is a type, and it is possible to declare the variable pointer to the function
TFunc func_ptr; // pointer to the function
//--- declare the functions corresponding to the TFunc description
int sub(int x,int y) { return(x-y); } // subtract one number from another
int add(int x,int y) { return(x+y); } // addition of two numbers
int neg(int x)       { return(~x); } // invert bits in the variable
//--- the func_ptr variable may store the function address to declare it later
func_ptr=sub;
Print(func_ptr(10,5));
func_ptr=add;
Print(func_ptr(10,5));
func_ptr=neg;           // error: neg does not have int (int,int) type
Print(func_ptr(10));    // error: two parameters needed
```

In this example, the *func_ptr* variable may receive the *sub* and *add* functions since they have two inputs each of `int` type as defined in the *TFunc* pointer to the function. On the contrary, the *neg* function cannot be assigned to the *func_ptr* pointer since its signature is different.

Arranging event models in the user interface

Pointers to functions allow you to easily create processing of events when creating a user interface. Let's use an example from the [CButton](#) section to show how to create buttons and add the functions for handling pressing to them. First, define a pointer to the *TAction* function to be called by pressing the button and create three functions according to the *TAction description*.


```

//--- create a custom function type
typedef int (*TAction) (string,int);
//+-----+
//|  Open the file                               |
//+-----+
int Open(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(1);
}
//+-----+
//|  Save the file                               |
//+-----+
int Save(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(2);
}
//+-----+
//|  Close the file                              |
//+-----+
int Close(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(3);
}

```

Then, create the MyButton class from [CButton](#), where we should add the *TAction* pointer to the function.

```

//+-----+
//|  Create the button class with the events processing function      |
//+-----+
class MyButton: public CButton
{
private:
    TAction      m_action;           // chart events handler
public:
    MyButton(void) {}
    ~MyButton(void) {}

    //--- constructor specifying the button text and the pointer to the events handling
    MyButton(string text, TAction act)
    {
        Text(text);
        m_action=act;
    }

    //--- set the custom function called from the OnEvent() events handler
    void          SetAction(TAction act) {m_action=act;}
}

```



```

//--- standard chart events handler
virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const
{
    if(m_action!=NULL && lparam==Id())
    {
        //--- call the custom m_action() handler
        m_action(sparam, (int)lparam);
        return(true);
    }
    else
        //--- return the result of calling the handler from the CButton parent class
        return(CButton::OnEvent(id,lparam,dparam,sparam));
}
};

```

Create the CControlsDialog derivative class from [CAppDialog](#), add the *m_buttons* array to it for storing the buttons of the *MyButton* type, as well as the *AddButton(MyButton &button)* and *CreateButtons()* methods.

```

//+-----+
//| CControlsDialog class                                |
//| Objective: graphical panel for managing the application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CArrayObj      m_buttons;                // button array
public:
    CControlsDialog(void) {} ;
    ~CControlsDialog(void) {} ;

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

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

```



```

//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowa
#define INDENT_TOP            (11)      // indent from top (with allowa
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)     // size by X coordinate
#define BUTTON_HEIGHT         (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//+-----+
//| Create and add buttons to the CControlsDialog panel |
//+-----+
bool CControlsDialog::CreateButtons(void)
{
//--- calculate buttons coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2;
    int y2=y1+BUTTON_HEIGHT;
//--- add buttons objects together with pointers to functions
    AddButton(new MyButton("Open",Open));
    AddButton(new MyButton("Save",Save));
    AddButton(new MyButton("Close",Close));
//--- create the buttons graphically
    for(int i=0;i<m_buttons.Total();i++)
    {
        MyButton *b=(MyButton*)m_buttons.At(i);
        x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
        x2=x1+BUTTON_WIDTH;
        if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
            PrintFormat("Failed to create button %s %d",b.Text(),i);
            return(false);
        }
//--- add each button to the CControlsDialog container
        if(!Add(b))
            return(false);
    }
//--- succeed
    return(true);
}

```

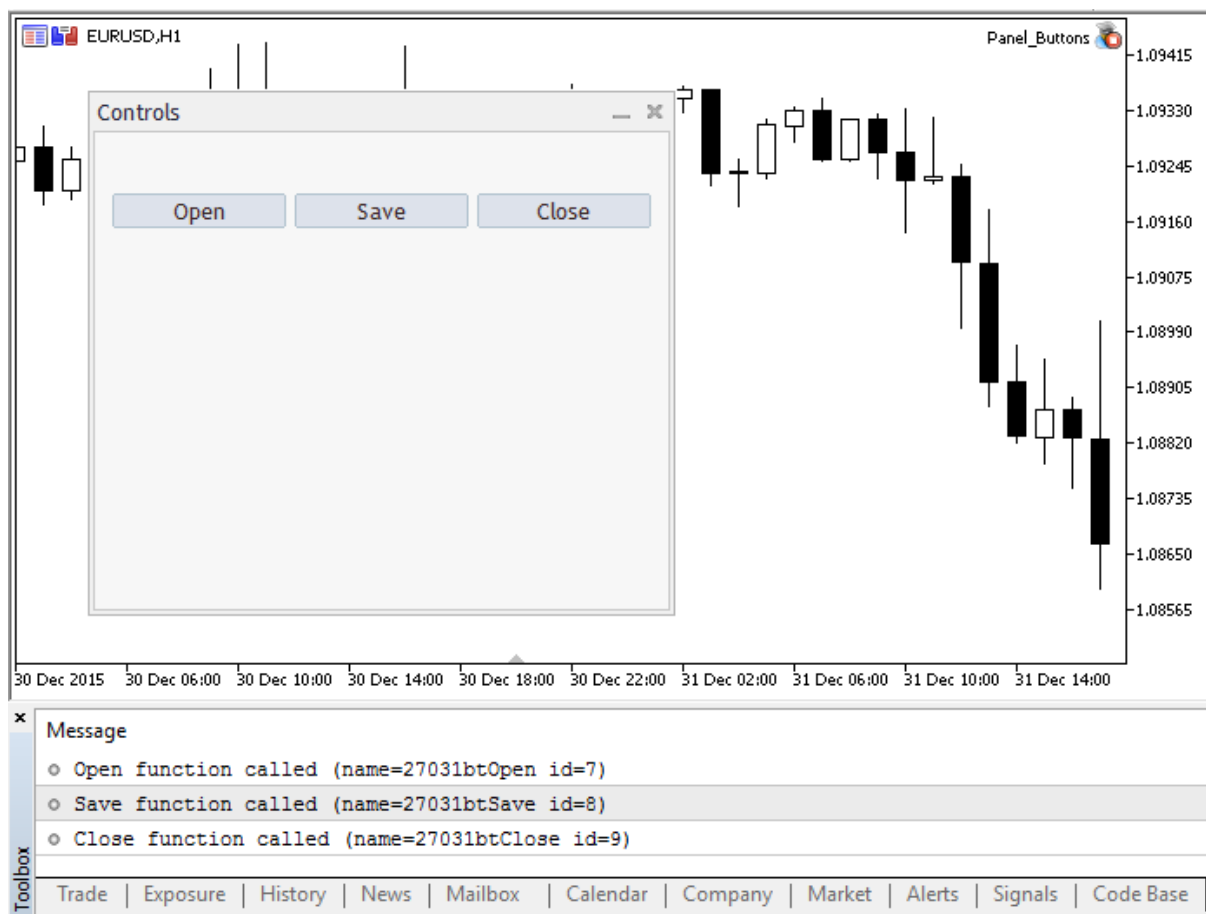
Now, we can develop the program using the CControlsDialog control panel having 3 buttons: Open, Save and Close. When clicking a button, the appropriate function in the form of the *TAction* pointer is called.


```

//--- declare the object on the global level to automatically create it when launching
CControlsDialog MyDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- now, create the object on the chart
    if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- launch the application
    MyDialog.Run();
//--- application successfully initialized
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- destroy dialog
    MyDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
//--- call the handler from the parent class (here it is CAppDialog) for the chart event
    MyDialog.ChartEvent(id,lparam,dparam,sparam);
}

```

The launched application's appearance and button clicking results are provided on the screenshot.



The full source code of the program

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The panel with several CButton buttons"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>

//+-----+
//| defines |
//+-----+

//--- indents and gaps
#define INDENT_LEFT      (11)      // indent from left (with allowa
#define INDENT_TOP       (11)      // indent from top (with allowa
#define CONTROLS_GAP_X   (5)       // gap by X coordinate
#define CONTROLS_GAP_Y   (5)       // gap by Y coordinate
```



```

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

//--- create the custom function type
typedef int(*TAction)(string,int);
//+-----+
//|  Open the file                                     |
//+-----+
int Open(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(1);
}
//+-----+
//|  Save the file                                     |
//+-----+
int Save(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(2);
}
//+-----+
//|  Close the file                                    |
//+-----+
int Close(string name,int id)
{
    PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
    return(3);
}
//+-----+
//| Create the button class with the events processing function |
//+-----+
class MyButton: public CButton
{
private:
    TAction      m_action;                // chart events handler
public:
    MyButton(void) {}
    ~MyButton(void) {}

    //--- constructor specifying the button text and the pointer to the events handling
    MyButton(string text,TAction act)
    {
        Text(text);
        m_action=act;
    }

    //--- set the custom function called from the OnEvent() events handler

```



```

void          SetAction(TAction act){m_action=act;}
//--- standard chart events handler
virtual bool   OnEvent(const int id,const long &lparam,const double &dparam,const
{
    if(m_action!=NULL && lparam==Id())
    {
        //--- call the custom handler
        m_action(sparam, (int)lparam);
        return(true);
    }
    else
        //--- return the result of calling the handler from the CButton parent class
        return(CButton::OnEvent(id,lparam,dparam,sparam));
}
};

//+-----+
//| CControlsDialog class |
//| Objective: graphical panel for managing the application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CArrayObj      m_buttons;          // button array
public:
        CControlsDialog(void){};
        ~CControlsDialog(void){};

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

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

//+-----+
//| Create and add buttons to the CControlsDialog panel |
//+-----+
bool CControlsDialog::CreateButtons(void)
{

```



```

//--- calculate buttons coordinates
int x1=INDENT_LEFT;
int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
int x2;
int y2=y1+BUTTON_HEIGHT;
//--- add buttons objects together with pointers to functions
AddButton(new MyButton("Open",Open));
AddButton(new MyButton("Save",Save));
AddButton(new MyButton("Close",Close));
//--- create the buttons graphically
for(int i=0;i<m_buttons.Total();i++)
{
    MyButton *b=(MyButton*)m_buttons.At(i);
    x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
    x2=x1+BUTTON_WIDTH;
    if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
    {
        PrintFormat("Failed to create button %s %d",b.Text(),i);
        return(false);
    }
    //--- add each button to the CControlsDialog container
    if(!Add(b))
        return(false);
}
//--- succeed
return(true);
}

//--- declare the object on the global level to automatically create it when launching
CControlsDialog MyDialog;

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- now, create the object on the chart
    if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- launch the application
    MyDialog.Run();
    //--- application successfully initialized
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- destroy dialog
    MyDialog.Destroy(reason);
}

```



```
    }  
    //+-----+  
    //| Expert chart event function |  
    //+-----+  
    void OnChartEvent(const int id,          // event ID  
                      const long& lparam,    // event parameter of the long type  
                      const double& dparam,  // event parameter of the double type  
                      const string& sparam) // event parameter of the string type  
        //--- call the handler from the parent class (here it is CAppDialog) for the chart event  
        MyDialog.ChartEvent(id, lparam, dparam, sparam);  
    }
```

See also

[Variables](#), [Functions](#)

Object Pointers

In MQL5, there is a possibility to dynamically create objects of complex type. This is done by the [new operator](#), which returns a descriptor of the created object. Descriptor is 8 bytes large. Syntactically, object descriptors in MQL5 are similar to pointers in C++.

Examples:

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

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

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

    //-- initializes object of type Foo
    void         Setup(string name)
    {
        m_name=name;
        s_counter++;
        m_id=s_counter;
    }
};

int Foo::s_counter=0;
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
    //-- declare an object as variable with its automatic creation
    Foo fool;
    //-- variant of passing an object by reference
    PrintObject(fool);

    //-- declare a pointer to an object and create it using the 'new' operator
    Foo *foo2=new Foo("foo2");
    //-- variant of passing a pointer to an object by reference
    PrintObject(foo2); // pointer to an object is converted automatically by compiler

    //-- declare an array of objects of type Foo
    Foo foo_objects[5];
```



```

//--- variant of passing an array of objects
PrintObjectsArray(foo_objects); // separate function for passing an array of objects

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

//--- variant of passing an array of pointers
PrintPointersArray(foo_pointers); // separate function for passing an array of pointers

//--- it is obligatory to delete objects created as pointers before termination
delete(foo2);

//--- delete array of pointers
int size=ArraySize(foo_pointers);
for(int i=0;i<5;i++)
    delete(foo_pointers[i]);
//---
}

//+-----+
//| Objects are always passed by reference |
//+-----+
void PrintObject(Foo &object)
{
    Print(__FUNCTION__, ": ", object.m_id, " Object name=", object.m_name);
}

//+-----+
//| Passing an array of objects |
//+-----+
void PrintObjectsArray(Foo &objects[])
{
    int size=ArraySize(objects);
    for(int i=0;i<size;i++)
    {
        PrintObject(objects[i]);
    }
}

//+-----+
//| Passing an array of pointers to object |
//+-----+
void PrintPointersArray(Foo* &objects[])
{
    int size=ArraySize(objects);
    for(int i=0;i<size;i++)
    {
        PrintObject(objects[i]);
    }
}

```



```
//+-----+
```

See also

[Variables](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

References: Modifier & and Keyword this

Passing Parameters by Reference

In MQL5 parameters of [simple](#) types can be passed both by value and by reference, while parameters of [compound](#) types are always passed by reference. To inform the compiler that a parameter must be passed by reference, the ampersand character & is added before the parameter name.

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

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

Example

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

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

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

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

Keyword this

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

The reserved word **this** is intended for obtaining the reference of the object to itself, which is available inside class or structure methods. **this** always references to the object, in the method of which it is used, and the expression [GetPointer](#)(this) gives the pointer of the object, whose member is the function, in which call of GetPointer() is performed. In MQL5 functions can't return objects, but they can return the object pointer.

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

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

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

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

//+-----+
//| returns its own pointer |
//+-----+
CDemoClass *CDemoClass::getDemoClass(void)
{
    return(GetPointer(this));
}
```

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

See also

[Object Pointers](#), [Creating and Deleting Objects](#), [Visibility Scope and Lifetime of Variables](#)

Operations and Expressions

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

+ - * / %	Symbols of arithmetic operations
&&	Symbols of logical operations
= += *=	Characters assignment operators

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

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

This section contains the description of the following topics:

- [Expressions](#)
- [Arithmetic Operations](#)
- [Assignment Operations](#)
- [Operations of Relation](#)
- [Boolean Operations](#)
- [Bitwise Operations](#)
- [Other Operations](#)
- [Priorities and Operations Order](#)

Expressions

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

Examples:

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

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

See also

[Precedence Rules](#)

Arithmetic Operations

Arithmetic operations include additive and multiplicative operations:

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

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

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

Important Notice

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

Computational problems may occur while moving the above expression from one programming environment to another one (for example, from Borland C++ to MQL5). In general, the order of computations depends on the compiler implementation. In practice, there are two ways to implement the post-decrement (post-increment):

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

Currently the first way of post-decrement (post-increment) calculation is implemented in MQL5. But even knowing this peculiarity, it is not recommended to experiment with its use.

Examples:

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

See also

[Precedence Rules](#)

Assignment Operations

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

Assigning the value of x to the y variable	<code>y = x;</code>
--	---------------------

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

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

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

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

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

<code>y=x=3;</code>

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

See also

[Precedence Rules](#)

Operations of Relation

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

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

True if a is equal to b	<code>a == b;</code>
True if a is not equal to b	<code>a != b;</code>
True if a is less than b	<code>a < b;</code>
True if a is greater than b	<code>a > b;</code>
True if a is less than or equal to b	<code>a <= b;</code>
True if a is greater than or equal to b	<code>a >= b;</code>

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

Example:

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

void OnStart()
{
    double first=0.3;
    double second=3.0;
    double third=second-2.7;
    if(first!=third)
    {
        if(CompareDoubles(first, third))
            printf("%.16f and %.16f are equal", first, third);
    }
}

// Result: 0.3000000000000000 0.2999999999999998 are equal
```

See also

[Precedence Rules](#)

Boolean Operations

Logical Negation NOT (!)

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

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

Logical Operation OR (||)

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

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

Logical Operation AND (&&)

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

Brief Estimate of Boolean Operations

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

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //-- the first example of the brief estimate
    if(func_false() && func_true())
    {
        Print("Operation &&: You will never see this expression");
    }
    else
    {
        Print("Operation &&: Result of the first expression is false, so the second wasn't");
    }
    //-- the second example of the brief estimate
    if(!func_false() || !func_true())
    {
        Print("Operation ||: Result of the first expression is true, so the second wasn't");
    }
    else
    {
        Print("Operation ||: You will never see this expression");
    }
}
//+-----+
```



```
///| the function always returns false |
//+-----+
bool func_false()
{
    Print("Function func_false()");
    return(false);
}
//+-----+
///| the function always returns true |
//+-----+
bool func_true()
{
    Print("Function func_true()");
    return(true);
}
```

See also[Precedence Rules](#)

Bitwise Operations

Complement to One

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

```
b = ~n;
```

Example:

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

Right Shift

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

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

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

Example:

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

Left Shift

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

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

Example:

```
char a='a',b='b';  
Print("Before: a = ",a, " b = ",b);  
//--- shift to the left  
b=a<<1;  
Print("After: a = ",a, " b = ",b);
```



```
// The result will be:
// Before:  a = 97   b = 98
// After:   a = 97   b = -62
```

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

Bitwise AND Operation

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

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

Example:

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

Bitwise OR Operation

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

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

Example:

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

Bitwise Exclusive Operation OR

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

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

Example:


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

Bitwise operations are performed with [integers](#) only.

See also

[Precedence Rules](#)

Other operations

Indexing ([])

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

Example:

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

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

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

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

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

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

Example:

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

Comma Operation (,)

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

Example:


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

Dot Operator (.)

For the direct [access to the public members](#) of structures and classes the dot operation is used. Syntax:

```
Variable_name_of_structure_type.Member_name
```

Example:

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

Scope Resolution Operation (::)

Each function in a mql5 program has its own execution scope. For example, the [Print\(\)](#) system function is performed in a global scope. [Imported](#) functions are called in the scope of the corresponding import. Method functions of [classes](#) have the scope of the corresponding class. The syntax of the scope resolution operation is as follows:

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

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

The scope resolution operation is also used to [define function](#)-class member.

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

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

1. Class methods. If no function with the specified name is set in the class, move to the next level.
2. MQL5 functions. If the language does not have such a function, move to the next level.
3. User defined global functions. If no function with the specified name is found, move to the next level.

4. Imported functions. If no function with the specified name is found, the compiler returns an error.

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

Example:

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

class CCheckContext
{
    int          m_id;
public:
                CCheckContext() { m_id=1234; }
protected:
    int          GetLastError() { return(m_id); }
};
class CCheckContext2 : public CCheckContext
{
    int          m_id2;
public:
                CCheckContext2() { m_id2=5678; }

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


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

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

Example:

```
sizeof (expression)
```

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

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

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

Example:

```
struct myStruct
{
    char    h;
    int     b;
    double  f;
} str;

Print("sizeof(str) = ", sizeof(str));
Print("sizeof(myStruct) = ", sizeof(myStruct));
```

The size is calculated at the compilation stage.

See also

[Precedence Rules](#)

Precedence Rules

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

Attention: Precedence of operations in the MQL5 language corresponds to the priority adopted in C++, and differs from the priority given in the MQL4 language.

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

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

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

Operators

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

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

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

Example:

```
if (Month() == 12)
```



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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Compound Operator

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

Example:

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Expression Operator

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

Assignment Operator

Identifier = expression;

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

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

Function Calling Operator

Function_name (argument1,..., argumentN);

```
FileClose(file);
```

Empty Operator

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Return Operator

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

Example:

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

In functions with the [void](#) return type, the [return](#) operator without expression must be used:

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

The right brace of the function means implicit execution of the [return](#) operator without expression.

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

If-Else Conditional Operator

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

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

If the expression is true, operator1 is executed and control is given to the operator that follows operator2 (operator2 is not executed). If the expression is false, operator2 is executed.

The **else** part of the **if** operator can be omitted. Thus, a divergence may appear in nested **if** operators with omitted **else** part. In this case, **else** addresses to the nearest previous **if** operator in the same block that has no **else** part.

Examples:

```
//--- The else part refers to the second if operator:
if(x>1)
    if(y==2) z=5;
else      z=6;
//--- The else part refers to the first if operator:
if(x>1)
{
    if(y==2) z=5;
}
else      z=6;
//--- Nested operators
if(x=='a')
{
    y=1;
}
else if(x=='b')
{
    y=2;
    z=3;
}
else if(x=='c')
{
    y=4;
}
else Print("ERROR");
```

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Ternary Operator ?:

The general form of the ternary operator is as follows:

```
expression1 ? expression2 : expression3
```

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

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

```

//--- normalize difference between open and close prices for a day range
double true_range = (High==Low)?0:(Close-Open)/(High-Low);

```

This entry is equivalent to the following:

```

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

```

Operator Use Restrictions

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

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

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

- a) Types must be identical or one should be [derived](#) from the other one.
- b) If types are not identical (inheritance), then the child is implicitly cast to the parent, i.e. the operator will be of the parent type.
- c) Do not mix object and the pointer - both expressions must be either objects or [pointers](#). [NULL](#) can be used for the pointer.

Note

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

Example:


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

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

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

    func(Expression1?Expression2:Expression3); // warning on implicit casting to string
    func(!Expression1?Expression2:Expression3); // warning on implicit casting to string
}

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Switch Operator

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

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

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

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

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

Examples:

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

//--- Second example
string res="";
int i=0;
switch(i)
{
    case 1:
        res=i;break;
```



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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

While Loop Operator

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

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

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

Note

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

Example:

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

For Loop Operator

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

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

Expression1 describes the loop initialization. Expression2 checks the conditions of the loop termination. If it is true, the loop body **for** is executed. The loop repeats expression2 until it becomes false. If it is false, the loop is terminated, and control is given to the next operator. Expression3 is calculated after each iteration.

The **for** operator is equivalent to the following succession of operators:

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

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

Note

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

Examples:

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Loop Operator do while

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

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

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

Note

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

Example:

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Break Operator

The **break** operator terminates the execution of the nearest nested outward [switch](#), [while](#), [do-while](#) or [for](#) operator. The control is passed to the operator that follows the terminated one. One of the purposes of this operator is to finish the looping execution when a certain value is assigned to a variable.

Example:

```
//--- searching for the first zero element
for(i=0;i<array_size;i++)
    if(array[i]==0)
        break;
```

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Continue Operator

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

Example:

```
//--- Sum of all nonzero elements
int func(int array[])
{
    int array_size=ArraySize(array);
    int sum=0;
    for(int i=0;i<array_size; i++)
    {
        if(a[i]==0) continue;
        sum+=a[i];
    }
    return(sum);
}
```

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Object Create Operator new

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

The new operator can be applied only to [class](#) objects. It can't be applied to structures.

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

Example:

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

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

An object created with the new operator must be explicitly removed using the [delete](#) operator.

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Object Delete Operator delete

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

Example:

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

See also

[Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Functions

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

```
function_header
{
    instructions
}
```

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

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

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

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

Example:

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

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

A function that returns no value should be described as that of [void](#) type.

Example:

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


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

Example:

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

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

Example of incorrect declaration:

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

See also

[Overload](#), [Virtual Functions](#), [Polymorphism](#)

Function Call

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

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

Arguments ([formal parameters](#)) are passed by value, i.e. each expression x_1, \dots, x_n is calculated, and the value is passed to the function. The order of expressions calculation and the order of values loading are not guaranteed. During the execution, the system checks the number and type of arguments passed to the function. Such way of addressing to the function is called a value call.

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

Examples:

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

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

Examples:

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

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

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

See also

[Overload](#), [Virtual Functions](#), [Polymorphism](#)

Passing Parameters

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

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

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

```
//+-----+
//| Passing parameters by reference |
//+-----+
double SecondMethod(int &i,int &j)
{
    double res;
//---
    i*=2;
    j/=2;
    res=i+j;
```



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

MQL5 uses both methods, with one exception: arrays, structure type variables and class objects are always passed by reference. In order to avoid changes in actual parameters (arguments passed at function call) use the access specifier [const](#). When trying to change the contents of a variable declared with the *const* specifier, the compiler will generate an error.

Note

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

Example:

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

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

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



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

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

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

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

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

See also

[Visibility Scope and Lifetime of Variables](#), [Overload](#), [Virtual Functions](#), [Polymorphism](#)

Function Overloading

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

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

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

Each function contains the message output via the [Print\(\)](#) function;

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


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

Example:

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

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

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

Below is a matching algorithm for each argument.

Algorithm of Choosing an Overloaded Function

1. Use strict matching (if possible).
2. Try standard type increase.
3. Try standard typecasting.

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

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

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

Examples of overloaded functions in MQL5 can be seen in the example of [ArrayInitialize\(\)](#) functions.

Function overloading rules apply to [overload of class methods](#).

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

Example:

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

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

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

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

See also

[Overload](#), [Virtual Functions](#), [Polymorphism](#)

Operation Overloading

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

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

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

For example, consider complex numbers, which consist of real and imaginary parts. They are widely used in mathematics. The MQL5 language has no data type to represent complex numbers, but it is possible to create a new data type in the form of a [structure or class](#). Declare the complex structure and define four methods that implement four arithmetic operations:

```
//+-----+
//| A structure for operations with complex numbers |
//+-----+
struct complex
{
    double      re; // Real part
    double      im; // Imaginary part
    //--- Constructors
    complex():re(0.0),im(0.0) { }
    complex(const double r):re(r),im(0.0) { }
    complex(const double r,const double i):re(r),im(i) { }
    complex(const complex &o):re(o.re),im(o.im) { }

    //--- Arithmetic operations
    complex      Add(const complex &l,const complex &r) const; // Addition
    complex      Sub(const complex &l,const complex &r) const; // Subtraction
    complex      Mul(const complex &l,const complex &r) const; // Multiplication
    complex      Div(const complex &l,const complex &r) const; // Division
};
```

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

For example:

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



```

z=a.Add(a,b);
PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- Multiply two numbers
z=a.Mul(a,b);
PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- Divide two numbers
z=a.Div(a,b);
PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
}

```

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

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

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

```

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

```

The full example of the script:

```

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

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

```



```

    PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
}
//+-----+
//| A structure for operations with complex numbers |
//+-----+
struct complex
{
    double      re; // Real part
    double      im; // Imaginary part
    //--- Constructors
        complex():re(0.0),im(0.0) { }
        complex(const double r):re(r),im(0.0) { }
        complex(const double r,const double i):re(r),im(i) { }
        complex(const complex &o):re(o.re),im(o.im) { }

    //--- Arithmetic operations
    complex      Add(const complex &l,const complex &r) const; // Addition
    complex      Sub(const complex &l,const complex &r) const; // Subtraction
    complex      Mul(const complex &l,const complex &r) const; // Multiplication
    complex      Div(const complex &l,const complex &r) const; // Division
    //--- Binary operators
    complex operator+(const complex &r) const { return(Add(this,r)); }
    complex operator-(const complex &r) const { return(Sub(this,r)); }
    complex operator*(const complex &r) const { return(Mul(this,r)); }
    complex operator/(const complex &r) const { return(Div(this,r)); }
};
//+-----+
//| Addition |
//+-----+
complex complex::Add(const complex &l,const complex &r) const
{
    complex res;
    //---
    res.re=l.re+r.re;
    res.im=l.im+r.im;
    //--- Result
    return res;
}
//+-----+
//| Subtraction |
//+-----+
complex complex::Sub(const complex &l,const complex &r) const
{
    complex res;
    //---
    res.re=l.re-r.re;
    res.im=l.im-r.im;
    //--- Result
    return res;
}

```



```

    }
//+-----+
//| Multiplication |
//+-----+
complex complex::Mul(const complex &l,const complex &r) const
{
    complex res;
//---
    res.re=l.re*r.re-l.im*r.im;
    res.im=l.re*r.im+l.im*r.re;
//--- Result
    return res;
}
//+-----+
//| Division |
//+-----+
complex complex::Div(const complex &l,const complex &r) const
{
//--- Empty complex number
    complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- Check for zero
    if(r.re==0 && r.im==0)
    {
        Print(__FUNCTION__+": number is zero");
        return(res);
    }
//--- Auxiliary variables
    double e;
    double f;
//--- Selecting calculation variant
    if(MathAbs(r.im)<MathAbs(r.re))
    {
        e = r.im/r.re;
        f = r.re+r.im*e;
        res.re=(l.re+l.im*e)/f;
        res.im=(l.im-l.re*e)/f;
    }
    else
    {
        e = r.re/r.im;
        f = r.im+r.re*e;
        res.re=(l.im+l.re*e)/f;
        res.im=(-l.re+l.im*e)/f;
    }
//--- Result
    return res;
}

```


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

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

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

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



```

complex z=a/b;
PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//--- A complex number is equal to zero by default (in the default constructor re==0 &
complex zero;
Print("!zero=",!zero);
//--- Assign a negative value
zero=-z;
PrintFormat("z=%.2f+i*%.2f, zero=%.2f+i*%.2f",z.re,z.im, zero.re,zero.im);
PrintFormat("-zero=%.2f+i*%.2f",-zero.re,-zero.im);
//--- Check for zero once again
Print("!zero=",!zero);
//---
}

```

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

Overloading of the indexing operator allows to obtain the values of the arrays enclosed in an object, in a simple and familiar way, and it also contributes to a better readability of the source code. For example, we need to provide access to a symbol in the string at the specified position. A string in MQL5 is a separate type [string](#), which is not an array of symbols, but with the help of an overloaded indexing operation we can provide a simple and transparent work in the generated CString class:

```

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

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

```



```
//--- Show the result
Print(res);
}
```

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

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- Operations of addition and multiplication of matrices
    CMatrix A(3),B(3),C();
//--- Prepare an array for rows
    double a1[3]={1,2,3}, a2[3]={2,3,1}, a3[3]={3,1,2};
    double b1[3]={3,2,1}, b2[3]={1,3,2}, b3[3]={2,1,3};
//--- Fill the matrices
    A[0]=a1; A[1]=a2; A[2]=a3;
    B[0]=b1; B[1]=b2; B[2]=b3;
//--- Output the matrices in the Experts log
    Print("---- Elements of matrix A");
    Print(A.String());
    Print("---- Elements of matrix B");
    Print(B.String());

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

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

//--- Now we show how to get values in the style of dynamic arrays matrix[i][j]
    Print("Output the values of matrix C elementwise");
//--- Go through the matrix rows - CRow objects - in a loop
    for(int i=0;i<3;i++)
    {
        string com="| ";
        //--- Form rows from the matrix for the value
        for(int j=0;j<3;j++)
        {
```



```

        //--- Get the matrix element by the number of the row and column
        double element=C[i][j]; // [i] - Access to CRow in the array m_rows[] ,
                                // [j] - Overloaded operator of indexing in CRow
        com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
    }
    com+="|";
    //--- Output the values of the row
    Print(com);
}

}

//+-----+
//| Class "Row" |
//+-----+
class CRow
{
private:
    double          m_array[];
public:
    //--- Constructors and a destructor
        CRow(void)          { ArrayResize(m_array,0);      }
        CRow(const CRow &r) { this=r;                      }
        CRow(const double &array[]);
        ~CRow(void) {};

    //--- Number of elements in the row
    int          Size(void) const { return(ArraySize(m_array)); }
    //--- Returns a string with values
    string       String(void) const;
    //--- Indexing operator
    double       operator[](int i) const { return(m_array[i]); }
    //--- Assignment operators
    void         operator=(const double &array[]); // An array
    void         operator=(const CRow & r);        // Another CRow object
    double       operator*(const CRow &o);         // CRow object for multiplication
};

//+-----+
//| Constructor for initializing a row with an array |
//+-----+
void CRow::CRow(const double &array[])
{
    int size=ArraySize(array);
    //--- If the array is not empty
    if(size>0)
    {
        ArrayResize(m_array,size);
        //--- Fill with values
        for(int i=0;i<size;i++)
            m_array[i]=array[i];
    }
}

//---

```



```

    }
//+-----+
//| Assignment operation for the array |
//+-----+
void CRow::operator=(const double &array[])
{
    int size=ArraySize(array);
    if(size==0) return;
//--- Fill the array with values
    ArrayResize(m_array,size);
    for(int i=0;i<size;i++) m_array[i]=array[i];
//---
}
//+-----+
//| Assignment operation for CRow |
//+-----+
void CRow::operator=(const CRow &r)
{
    int size=r.Size();
    if(size==0) return;
//--- Fill the array with values
    ArrayResize(m_array,size);
    for(int i=0;i<size;i++) m_array[i]=r[i];
//---
}
//+-----+
//| Operator of multiplication by another row |
//+-----+
double CRow::operator*(const CRow &o)
{
    double res=0;
//--- Verifications
    int size=Size();
    if(size!=o.Size() || size==0)
    {
        Print(__FUNCSIG__,": Failed to multiply two matrices, their sizes are different");
        return(res);
    }
//--- Multiply arrays elementwise and add the products
    for(int i=0;i<size;i++)
        res+=m_array[i]*o[i];
//--- Result
    return(res);
}
//+-----+
//| Returns a formatted string representation |
//+-----+
string CRow::String(void) const
{

```



```

    string out="";
    //--- If the size of the array is greater than zero
    int size=ArraySize(m_array);
    //--- We work only with a non-zero number of array elements
    if(size>0)
    {
        out="{ ";
        for(int i=0;i<size;i++)
        {
            //--- Collect the values to a string
            out+=StringFormat(" %G;",m_array[i]);
        }
        out+=" }";
    }
    //--- Result
    return(out);
}

//+-----+
//| Class "Matrix"                                     |
//+-----+
class CMatrix
{
private:
    CRow          m_rows[];

public:
    //--- Constructors and a destructor
        CMatrix(void);
        CMatrix(int rows) { ArrayResize(m_rows,rows); }
        ~CMatrix(void){};

    //--- Get the matrix sizes
    int      Rows()      const { return(ArraySize(m_rows)); }
    int      Cols()      const { return(Rows()>0? m_rows[0].Size():0); }
    //--- Returns the value of the column in the form of a CRow row
    CRow      GetColumnAsRow(const int col_index) const;
    //--- Returns a string with the matrix values
    string      String(void) const;
    //--- The indexing operator returns a string by its number
    CRow *operator[](int i) const { return(GetPointer(m_rows[i])); }
    //--- Addition operator
    CMatrix      operator+(const CMatrix &m);
    //--- Multiplication operator
    CMatrix      operator*(const CMatrix &m);
    //--- Assignment operator
    CMatrix      *operator=(const CMatrix &m);
};

//+-----+
//| A default constructor, create an array of rows of zero size |
//+-----+

```



```

CMatrix::CMatrix(void)
{
//--- The zero number of rows in the matrix
    ArrayResize(m_rows,0);
//---
}
//+-----+
//| Returns the column value in the form of CRow |
//+-----+
CRow CMatrix::GetColumnAsRow(const int col_index) const
{
//--- A variable to get the values from the column
    CRow row();
//--- The number of rows in the matrix
    int rows=Rows();
//--- If the number of rows is greater than zero, execute the operation
    if(rows>0)
    {
        //--- An array to receive the values of the column with index col_index
        double array[];
        ArrayResize(array,rows);
        //--- Filling the array
        for(int i=0;i<rows;i++)
        {
            //--- Check the number of the column for row i - it may exceed the boundaries
            if(col_index>=this[i].Size())
            {
                Print(__FUNCSIG__,": Error! Column number ",col_index,"> row size ",i);
                break; // row will be uninitialized object
            }
            array[i]=this[i][col_index];
        }
        //--- Create a CRow row based on the array values
        row=array;
    }
//--- Result
    return(row);
}
//+-----+
//| Addition of two matrices |
//+-----+
CMatrix CMatrix::operator+(const CMatrix &m)
{
//--- The number of rows and columns in the passed matrix
    int cols=m.Cols();
    int rows=m.Rows();
//--- The matrix to receive the addition results
    CMatrix res(rows);
//--- The sizes of the matrix must match

```



```

    if(cols!=Cols() || rows!=Rows())
    {
        //--- Addition impossible
        Print(__FUNCSIG__, ": Failed to add two matrices, their sizes are different");
        return(res);
    }
//--- Auxiliary array
    double arr[];
    ArrayResize(arr,cols);
//--- Go through rows to add
    for(int i=0;i<rows;i++)
    {
        //--- Write the results of addition of matrix strings in the array
        for(int k=0;k<cols;k++)
        {
            arr[k]=this[i][k]+m[i][k];
        }
        //--- Place the array to the matrix row
        res[i]=arr;
    }
//--- return the result of addition of matrices
    return(res);
}
//+-----+
//| Multiplication of two matrices |
//+-----+
CMatrix CMatrix::operator*(const CMatrix &m)
{
//--- Number of columns of the first matrix, number of rows passed in the matrix
    int cols1=Cols();
    int rows2=m.Rows();
    int rows1=Rows();
    int cols2=m.Cols();
//--- Matrix to receive the addition result
    CMatrix res(rows1);
//--- Matrices should be coordinated
    if(cols1!=rows2)
    {
        //--- Multiplication impossible
        Print(__FUNCSIG__, ": Failed to multiply two matrices, format is not compatible '
            \"- number of columns in the first factor should be equal to the number of
        return(res);
    }
//--- Auxiliary array
    double arr[];
    ArrayResize(arr,cols1);
//--- Fill the rows in the multiplication matrix
    for(int i=0;i<rows1;i++)// Go through rows
    {

```



```

    //--- Reset the receiving array
    ArrayInitialize(arr,0);
    //--- Go through elements in the row
    for(int k=0;k<cols1;k++)
    {
        //--- Take values of column k of the matrix m in the for of CRow
        CRow column=m.GetColumnAsRow(k);
        //--- Multiply two rows and write the result of scalar multiplication of vectors
        arr[k]=this[i]*column;
    }
    //--- place array arr[] in the i-th row of the matrix
    res[i]=arr;
}

//--- Return the product of two matrices
return(res);
}

//+-----+
//| Assignment operation |
//+-----+
CMatrix *CMatrix::operator=(const CMatrix &m)
{
    //--- Find and set the number of rows
    int rows=m.Rows();
    ArrayResize(m_rows,rows);
    //--- Fill our rows with the values of rows of the passed matrix
    for(int i=0;i<rows;i++) this[i]=m[i];
    //---
    return(GetPointer(this));
}

//+-----+
//| String representation of the matrix |
//+-----+
string CMatrix::String(void) const
{
    string out="";
    int rows=Rows();
    //--- Form string by string
    for(int i=0;i<rows;i++)
    {
        out=out+this[i].String()+"\r\n";
    }
    //--- Result
    return(out);
}

```

See also

[Overloading](#), [Arithmetic Operations](#), [Function Overloading](#), [Precedence Rules](#)

Description of External Functions

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

Examples:

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

With the help of `import`, it is easy to describe functions that are called from external DLL or compiled EX5 libraries. EX5 libraries are compiled ex5 files, which have the [library](#) property. Only function described with [the export modifier](#) can be imported from EX5 libraries.

Please keep in mind that DLL and EX5 libraries should have different names (regardless of the directories they are located in) if they are imported together. All imported functions have the scope resolution corresponding to the library's "file name".

Example:

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

class CFoo
{
public:
    int      GetLastError() { return(12345); }
    void      func()
    {
        Print(GetLastError());           // call of the class method
        Print(::GetLastError());         // call of the MQL5 function
        Print(kernel32::GetLastError()); // call of the DLL library function from kernel32.dll
        Print(lib::GetLastError());      // call of the EX5 library function from lib.ex5
    }
};

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


See also

[Overload](#), [Virtual Functions](#), [Polymorphism](#)

Exporting Functions

A function declared in a mql5 program with the *export* postmodifier can be used in another mql5 program. Such a function is called exportable, and it can be called from other programs after compilation.

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

This modifier orders the compiler to add the function into the table of EX5 functions exported by this ex5 file. Only function with such a modifier are accessible ("visible") from other mql5 programs.

The [library](#) property tells the compiler that the EX5-file will be a library, and the compiler will show it in the header of EX5.

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

See also

[Overload](#), [Virtual Functions](#), [Polymorphism](#)

Event Handling Functions

The MQL5 language provides processing of some [predefined events](#). Functions for handling these events must be defined in a MQL5 program; function name, return type, composition of parameters (if there are any) and their types must strictly conform to the description of the event handler function.

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

OnStart

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

```
void OnStart();
```

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

OnInit

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

```
void OnInit();
```

The Init event is generated immediately after an Expert Advisor or an indicator is downloaded; this event is not generated for scripts. The OnInit() function is used for initialization. If OnInit() has the int type of the return value, the non-zero return code means unsuccessful initialization, and it generates the [Deinit](#) event with the code of deinitialization reason [REASON_INITFAILED](#).

To optimize input parameters of an Expert Advisor, it is recommended to use values of the [ENUM_INIT_RETCODE](#) enumeration as the return code. These values are used for organizing the course of optimization, including the selection of the most appropriate [testing agents](#). During initialization of an Expert Advisor before the start of testing you can request information about the configuration and resources of an agent (the number of cores, amount of free memory, etc.) using the [TerminalInfoInteger\(\)](#) function. Based on the information obtained, you can either allow to use this testing agent, or reject using it during the optimization of this Expert Advisor.

ENUM_INIT_RETCODE

Identifier	Description
INIT_SUCCEEDED	Successful initialization, testing of the Expert Advisor can be continued. This code means the same as a null value - the Expert Advisor has been successfully initialized in the tester.
INIT_FAILED	Initialization failed; there is no point in continuing testing because of fatal errors. For example, failed to create an indicator that is required for the work of the Expert Advisor.

	This return value means the same as a value other than zero - initialization of the Expert Advisor in the tester failed.
INIT_PARAMETERS_INCORRECT	This value means the incorrect set of input parameters. The result string containing this return code is highlighted in red in the general optimization table. Testing for the given set of parameters of the Expert Advisor will not be executed, the agent is free to receive a new task. Upon receiving this value, the strategy tester will reliably not pass this task to other agents for retry.
INIT_AGENT_NOT_SUITABLE	No errors during initialization, but for some reason the agent is not suitable for testing. For example, not enough memory, no OpenCL support , etc. After the return of this code, the agent will not receive tasks until the end of this optimization .

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

OnDeinit

The OnDeinit() function is called during deinitialization and is the [Deinit](#) event handler. It must be declared as the **void** type and should have one parameter of the **const int** type, which contains [the code of deinitialization reason](#). If a different type is declared, the compiler will generate a warning, but the function will not be called. For scripts the Deinit event is not generated and therefore the OnDeinit() function can't be used in scripts.

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

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

- before reinitialization due to the change of a symbol or chart period, to which the mql5 program is attached;
- before reinitialization due to the change of [input parameters](#);
- before unloading the mql5 program.

OnTick

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

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

```
void OnTick();
```


OnTimer

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

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

```
void OnTimer();
```

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

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as the mql5 program stops operating, the timer is destroyed forcibly, if it was created but hasn't been disabled by the [EventKillTimer\(\)](#) function.

OnTrade

The function is called when the [Trade](#) event occurs, which appears when you change the list of [placed orders](#) and [open positions](#), [the history of orders](#) and [history of deals](#). When a trade activity is performed (pending order opening, position opening/closing, stops setting, pending order triggering, etc.) the history of orders and deals and/or list of positions and current orders is changed accordingly.

```
void OnTrade();
```

Users must independently implement in the code the verification of a trade account state when such an event is received (if this is required by the trade strategy conditions). If the `OrderSend()` function call has been completed successfully and returned a value of true, this means that the trading server has put the order into the queue for execution and assigned a ticket number to it. As soon as the server processes this order, the Trade event will be generated. And if a user remembers the ticket value, he/she will be able to find out what happened to the order using this value during `OnTrade()` event handling.

OnTradeTransaction

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using [OrderSend](#) and [OrderSendAsync](#) functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;

- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions. Arrival of such a transaction at the terminal is a [TradeTransaction](#) event. It calls OnTradeTransaction handler

```
void OnTradeTransaction(
    const MqlTradeTransaction& trans,      // trade transaction structure
    const MqlTradeRequest& request,        // request structure
    const MqlTradeResult& result           // result structure
);
```

The handler contains three parameters:

- **trans** - this parameter gets [MqlTradeTransaction](#) structure describing a trade transaction applied to a trade account;
- **request** - this parameter gets [MqlTradeRequest](#) structure describing a trade request;
- **result** - this parameter gets [MqlTradeResult](#) structure describing a trade request execution result.

The last two **request** and **result** parameters are filled by values only for [TRADE_TRANSACTION_REQUEST](#) type transaction, data on transaction can be received from *type* parameter of **trans** variable. Note that in this case, *request_id* field in **result** variable contains ID of **request** [trade request](#), after the execution of which the [trade transaction](#) described in **trans** variable has been performed. Request ID allows to associate the performed action (OrderSend or OrderSendAsync functions call) with the result of this action sent to [OnTradeTransaction\(\)](#).

One trade request manually sent from the terminal or via [OrderSend\(\)](#)/[OrderSendAsync\(\)](#) functions can generate several consecutive transactions on the trade server. Priority of these transactions' arrival at the terminal is not guaranteed. Thus, you should not expect that one group of transactions will arrive after another one when developing your trading algorithm.

- All types of trade transactions are described in [ENUM_TRADE_TRANSACTION_TYPE](#) enumeration.
- MqlTradeTransaction structure describing a trade transaction is filled in different ways depending on a transaction type. For example, only type field (trade transaction type) must be analyzed for [TRADE_TRANSACTION_REQUEST](#) type transactions. The second and third parameters of OnTradeTransaction function (request and result) must be analyzed for additional data. For more information, see ["Structure of a Trade Transaction"](#).
- A trade transaction description does not deliver all available information concerning orders, deals and positions (e.g., comments). [OrderGet*](#), [HistoryOrderGet*](#), [HistoryDealGet*](#) and [PositionGet*](#) functions should be used to get extended information.

After applying trade transactions for a client account, they are consistently placed to the terminal trade transactions queue, from which they consistently sent to OnTradeTransaction entry point in order of arrival at the terminal.

When handling trade transactions by an Expert Advisor using OnTradeTransaction handler, the terminal continues handling newly arrived trade transactions. Therefore, the state of a trade account can change during OnTradeTransaction operation already. For example, while an MQL5 program handles an event of adding a new order, it may be executed, deleted from the list of the open ones and moved to the history. Further on, the application will be notified of these events.

Transactions queue length comprises 1024 elements. If OnTradeTransaction handles a new transaction for too long, the old ones in the queue may be superseded by the newer ones.

- Generally, there is no accurate ratio of the number of OnTrade and OnTradeTransaction calls. One OnTrade call corresponds to one or several OnTradeTransaction calls.
- OnTrade is called after appropriate OnTradeTransaction calls.

OnTester

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

```
double OnTester();
```

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

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

OnTesterInit

The OnTesterInit() function is the handler of the [TesterInit](#) event, which is automatically generated before the start of Expert Advisor optimization in the strategy tester. The function must be defined with the void type. It has no parameters:

```
void OnTesterInit();
```

With the start of optimization, an Expert Advisor with the OnTesterDeinit() or OnTesterPass() handler is automatically loaded in a separate terminal chart with the symbol and period specified in the tester, and receives the TesterInit event. The function is used for Expert Advisor initialization before the start of optimization for further [processing of optimization results](#).

OnTesterPass

The OnTesterPass() function is the handler of the [TesterPass](#) event, which is automatically generated when a frame is received during Expert Advisor optimization in the strategy tester. The function must be defined with the void type. It has no parameters:

```
void OnTesterPass();
```

An Expert Advisor with the OnTesterPass() handler is automatically loaded in a separate terminal chart with the symbol/period specified for testing, and gets TesterPass events when a frame is received during optimization. The function is used for dynamic handling of [optimization results](#) "on the spot" without waiting for its completion. Frames are added using the [FrameAdd\(\)](#) function, which can be called after the end of a single pass in the [OnTester\(\)](#) handler.

OnTesterDeinit

OnTesterDeinit() is the handler of the [TesterDeinit](#) event, which is automatically generated after the end of Expert Advisor optimization in the strategy tester. The function must be defined with the void type. It has no parameters:

```
void OnTesterDeinit();
```

An Expert Advisor with the TesterDeinit() handler is automatically loaded on a chart at the start of optimization, and receives TesterDeinit after its completion. The function is used for final processing of all [optimization results](#).

OnBookEvent

The OnBookEvent() function is the [BookEvent](#) handler. BookEvent is generated for Expert Advisors and indicators when Depth of Market changes. It must be of the void type and have one parameter of the string type:

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

To receive BookEvent events for any symbol, you just need to pre-subscribe to receive these events for this symbol using the [MarketBookAdd\(\)](#) function. In order to unsubscribe from receiving the BookEvent events for a particular symbol, call [MarketBookRelease\(\)](#).

Unlike other events, the BookEvent event is broadcast. This means that if one Expert Advisor subscribes to receiving BookEvent events using MarketBookAdd, all the other Experts Advisors that have the OnBookEvent() handler will receive this event. It is therefore necessary to analyze the name of the symbol, which is passed to the handler as the *const string& symbol* parameter.

OnChartEvent

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

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

- CHARTEVENT_CUSTOM_LAST – the last acceptable ID of a custom event (CHARTEVENT_CUSTOM +65535).

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

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

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

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of times the keystroke is repeated as a result of the user holding down the key)	The string value of a bit mask describing the status of keyboard buttons
Mouse events (if property CHART_EVENT_MOUSE_MOVE =true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
Event of graphical object creation (if CHART_EVENT_OBJECT_CREATE =true is set for the chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE =true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object

BJECT_DELETE =true is set for the chart)				
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit graphical object, in which text editing has completed
Event of chart Changes	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

OnCalculate

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

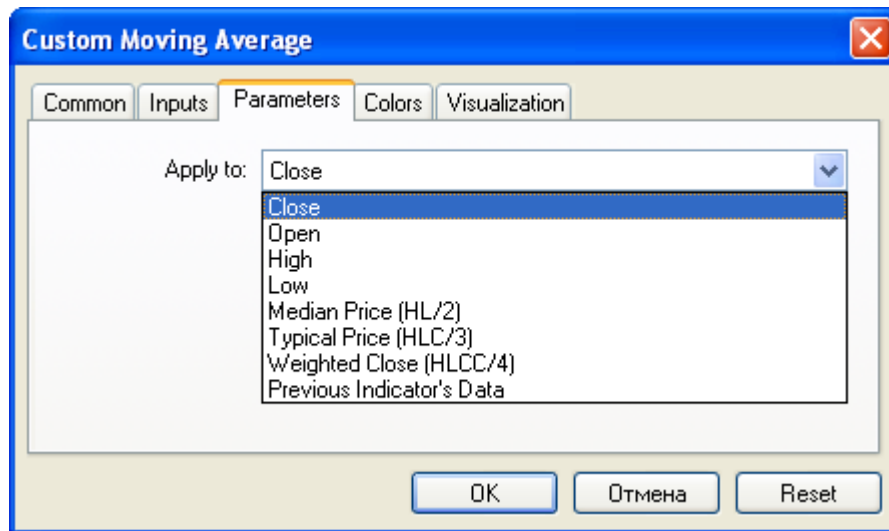
The OnCalculate() function must have a return type int. There are two possible definitions. Within one indicator you cannot use both versions of the function.

The first form is intended for those indicators that can be calculated on a single data buffer. An example of such an indicator is Custom Moving Average.

```
int OnCalculate (const int rates_total,      // size of the price[] array
                const int prev_calculated,  // bars handled on a previous call
                const int begin,            // where the significant data start from
                const double& price[]       // array to calculate
                );
```


As the price[] array, one of timeseries or a calculated buffer of some indicator can be passed. To determine the direction of indexing in the price[] array, call [ArrayGetAsSeries\(\)](#). In order not to depend on the default values, you must unconditionally call the [ArraySetAsSeries\(\)](#) function for those arrays, that are expected to work with.

Necessary time series or an indicator to be used as the price[] array can be selected by the user in the "Parameters" tab when starting the indicator. To do this, you should specify the necessary item in the drop-down list of "Apply to" field.



To receive values of a custom indicator from other mql5 programs, the [iCustom\(\)](#) function is used, which returns the indicator handle for subsequent operations. You can also specify the appropriate price[] array or the handle of another indicator. This parameter should be transmitted last in the list of input variables of the custom indicator.

Example:

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

In this example, the last parameter passed is the PRICE_TYPICAL value (from the [ENUM_APPLIED_PRICE](#) enumeration), which indicates that the custom indicator will be built on typical prices obtained as (High+Low+Close)/3. If this parameter is not specified, the indicator is built based on PRICE_CLOSE values, i.e. closing prices of each bar.

Another example that shows passing of the indicator handler as the last parameter to specify the price[] array, is given in the description of the [iCustom\(\)](#) function.

The second form is intended for all other indicators, in which more than one time series is used for calculations.

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

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

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

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

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

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

Note: if `OnCalculate` returns zero, then the indicator values are not shown in the DataWindow of the client terminal.

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

Indicator Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Line
#property indicator_label1 "Line"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDarkBlue
```



```

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

```

See also

[Running Programs](#), [Client Terminal Events](#), [Working with Events](#)

Variables

Declaring Variables

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

Simple types are:

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

Examples:

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

Complex or compound types:

Structures are composite data types, constructed using other types.

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

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

Arrays

Array is the indexed sequence of identical-type data:

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


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

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

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

Access Specifiers

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

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

Example:

```
int OnCalculate (const int rates_total,      // size of the price[] array
                const int prev_calculated,  // bars handled on a previous call
                const int begin,            // where the significant data start from
                const double& price[]       // array to calculate
                );
```

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

- [public](#) - allows unrestricted access to the variable or class method
- [protected](#) - allows access from methods of this class, as well as from methods of [publicly inherited](#) classes. Other access is impossible;
- `private` - allows access to variables and class methods only from methods of the same class.
- [virtual](#) - applies only to class methods (but not to methods of structures) and tells the compiler that this method should be placed in the table of virtual functions of the class.

Storage Classes

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

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#), [Static Members of a Class](#)

Local Variables

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

Example:

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

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

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

Example:

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

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

Example:

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



```
Inside for i = 0
Inside for i = 1
Inside for i = 2
Outside block i = 5
*/
```

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

Stack

In every MQL5 program, a special memory area called stack is allocated for storing local function variables that are created automatically. One stack is allocated for all functions, its default size for indicators is equal to 1Mb. In Expert Advisors and scripts, stack size can be managed using the [#property stacksize](#) compiler directive (which sets the stack size in bytes), a memory of 8Mb is allocated by default for the stack.

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

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

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

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

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Formal Parameters

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

Example:

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

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

Example:

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

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

Example:

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

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

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

Example:

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



```
z[i]=OrderOpenPrice();  
}  
x=i;  
y=calculated_tp;  
}
```

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

Maximum 64 parameters can be passed into a function.

See also

[Input Variables](#), [Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Static Variables

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

Example:

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

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

Static variables exist from the moment of program execution and are initialized only once before the specialized functions [OnInit\(\)](#) is called. If the initial values are not specified, variables of the static storage class are taking zero initial values.

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

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

Example:

```
int Counter()
{
    static int count;
    count++;
    if(count%100==0) Print("Function Counter has been called ",count," times");
    return count;
}

void OnStart()
{
    //---
    int c=345;
    for(int i=0;i<1000;i++)
    {
        int c=Counter();
    }
    Print("c =",c);
}
```

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#), [Static Class Members](#)

Global Variables

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

Example:

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

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

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

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

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

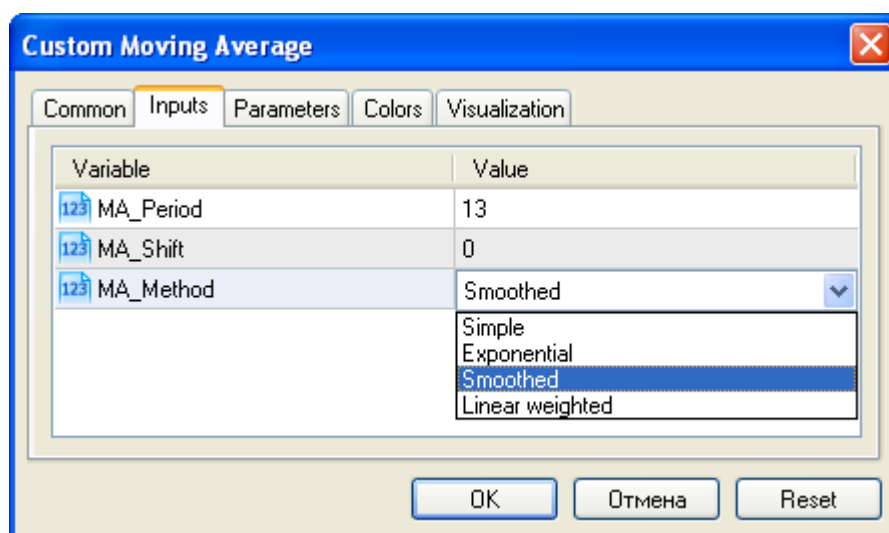
Input Variables

The `input` storage class defines the external variable. The `input` modifier is indicated before the data type. A variable with the input modifier can't be changed inside mql5-programs, such variables can be accessed for reading only. Values of input variables can be changed only by a user from the program properties window. External variables are always reinitialized immediately before the `OnInit()` is called.

Example:

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

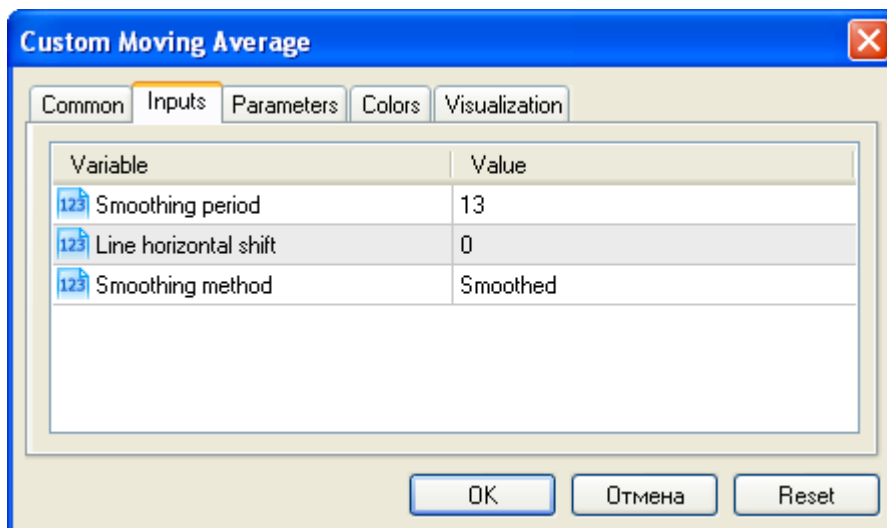
Input variables determine the input parameters of a program. They are available from the Properties window of a program.



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

Example:

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

Note: Arrays and variables of [complex types](#) can't act as input variables.

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

Passing Parameters When Calling Custom Indicators from MQL5 Programs

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

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

Enumerations as input Parameters

Not only built-in enumerations provided in MQL5, but also user defined variables can be used as input variables (input parameters for mql5 programs). For example, we can create the `dayOfWeek` enumeration, describing days of the week, and use the input variable to specify a particular day of the week, not as a number, but in a more common way.

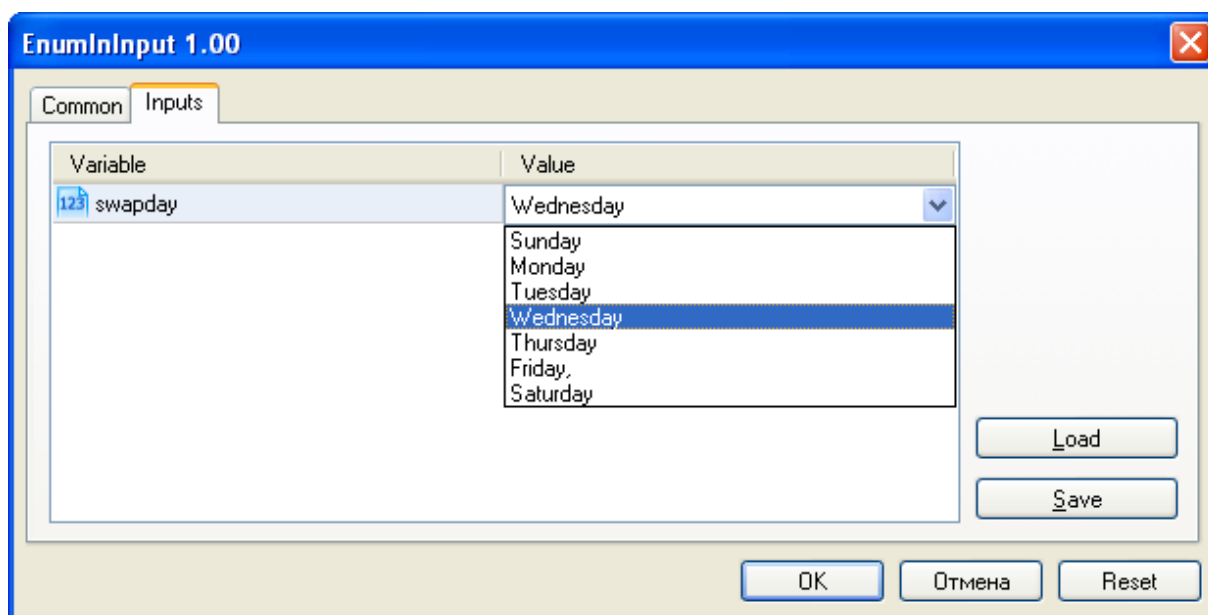
Example:

```
#property script_show_inputs
//--- day of week
enum dayOfWeek
{
    S=0,      // Sunday
    M=1,      // Monday
    T=2,      // Tuesday
    W=3,      // Wednesday
    Th=4,     // Thursday
}
```



```
Fr=5,    // Friday,
St=6,    // Saturday
};
//--- input parameters
input dayOfWeek swapday=W;
```

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



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

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

Variables with sinput Modifier

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

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


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

will be equivalent to the full declaration

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

The variable declared with `sinput` modifier is an input parameter of MQL5 program. The value of this parameter can be changed when launching the program. However, this variable is not used in the optimization of input parameters. In other words, its values are not enumerated when searching for the best set of parameters fitting a specified condition.

Variable	Value	Start	Step	Stop	Steps
<input type="checkbox"/> Number of layers	6				
<input checked="" type="checkbox"/> Neurons in a layer	30	30	1	300	271
<input checked="" type="checkbox"/> Number of bars to be analyzed	13	13	1	130	118
<input checked="" type="checkbox"/> Forecast horizon	2	2	1	20	19
<input type="checkbox"/> Network type	0	0	1	10	
					607582

Strategy Tester

Settings Inputs Agents Journal

For Help, press F1 Default

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

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

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

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

See also

[iCustom](#), [Enumerations](#), [Properties of Programs](#)

Extern Variables

The `extern` keyword is used for declaring variable identifiers as identifiers of the [static storage class](#) with global [lifetime](#). These variables exist from the start of the program and memory for them is allocated and initialized immediately after the start of the program.

You can create programs that consist of multiple source files; in this case a directive to the preprocessor `#include` is used. Variables declared as an extern with the same type and identifier can exist in different source files of one project.

When compiling the whole project, all the extern variables with the same type and an identifier are associated with one part of memory of the global variable pool. Extern variables are useful for separate compilation of source files. Extern variables can be initialized, but only once - existence of several initialized extern variables of the same type and with the same identifier is prohibited.

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Initialization of Variables

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

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

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

Examples:

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

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

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

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

Examples:

```
struct str3
{
    int          low_part;
    int          high_part;
};
struct str10
{
    str3         s3;
    double       d1[10];
    int          i3;
};
void OnStart()
{
    str10 s10_1={{1,0},{1.0,2.1,3.2,4.4,5.3,6.1,7.8,8.7,9.2,10.0},100};
    str10 s10_2={{1,0},{0},100};
```



```
str10 s10_3={{1,0},{1.0}};  
//---  
Print("1.  s10_1.d1[5] = ",s10_1.d1[5]);  
Print("2.  s10_2.d1[5] = ",s10_2.d1[5]);  
Print("3.  s10_3.d1[5] = ",s10_3.d1[5]);  
Print("4.  s10_3.d1[0] = ",s10_3.d1[0]);  
}
```

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

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Visibility Scope and Lifetime of Variables](#), [Creating and Deleting Objects](#)

Visibility Scope and Lifetime of Variables

There are two basic types of scope: [local](#) scope and [global](#) scope.

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

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

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

Example:

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

    for(i=limit; i<rates_total; i++)
    {
        sum=0;
```



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

Pay attention to the variable i, declared in line

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

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

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

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Creating and Deleting Objects](#)

Creating and Deleting Objects

After a MQL5 program is loaded for execution, memory is allocated to each variable according to its type. According to the access level, all variables are divided into two types - [global variables](#) and [local variables](#). According to the memory class, they can be [input parameters](#) of a MQL5 program, [static](#) and [automatic](#). If necessary, each variable is [initialized](#) by a corresponding value. After being used a variable is uninitialized and memory used by it is returned to the MQL5 executable system.

Initialization and Deinitialization of Global Variables

Global variables are initialized automatically right after a MQL5 program is loaded and before any of function is called. During initialization initial values are assigned to variables of [simple](#) types and a constructor (if there is any) is called for objects. [Input variables](#) are always declared at a global level, and are initialized by values set by a user in the dialog during the program start.

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

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

Initialization and Deinitialization of Local Variables

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

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

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

Initialization and Deinitialization of Objects Placed

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

Pointers can be declared on a local or global level; and they can be initialized by the empty value of [NULL](#) or by the value of the pointer of the same or [inherited](#) type. If the [new](#) operator is called for a

pointer declared on a local level, the *delete* operator for this pointer must be performed before exiting the level. Otherwise the pointer will be lost and the explicit deletion of the object will fail.

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

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

Thus dynamically placed objects are created only at the moment when the corresponding *new* operator is invoked, and are assuredly deleted either by the *delete* operator or automatically by the executing system of MQL5 during the program unloading. The order of declaration of pointers of dynamically created object doesn't influence the order of their initialization. The order of initialization and deinitialization is fully controlled by the programmer.

Dynamic memory allocation in MQL5

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

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

Memory manager releases memory back to the operating system immediately after exiting the following event handling functions: [OnInit\(\)](#), [OnDeinit\(\)](#), [OnStart\(\)](#), [OnTick\(\)](#), [OnCalculate\(\)](#), [OnTimer\(\)](#), [OnTrade\(\)](#), [OnTester\(\)](#), [OnTesterInit\(\)](#), [OnTesterPass\(\)](#), [OnTesterDeinit\(\)](#), [OnChartEvent\(\)](#), [OnBookEvent\(\)](#).

Brief Characteristics of Variables

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

	Global automatic variable	Local automatic variable	Dynamically created object
Initialization	right after a mql5 program is loaded	when the code line where it is declared is reached during execution	at the execution of the <i>new</i> operator
Initialization order	in the order of declaration	in the order of declaration	irrespective of the order of declaration
Deinitialization	before a mql5 program is unloaded	when execution exits the declaration block	when the <i>delete</i> operator is executed

			or before a mql5 program is unloaded
Deinitialization order	in the order opposite to the initialization order	in the order opposite to the initialization order	irrespective of the initialization order
Constructor call	at mql5 program loading	at initialization	at the execution of the <i>new</i> operator
Destructor call	at mql5 program unloading	when exiting the block where the variable was initialized	at the execution of the <i>delete</i> operator
Error logs	log message in the "Experts" journal about the attempt to delete an automatically created object	log message in the "Experts" journal about the attempt to delete an automatically created object	log message in the "Experts" journal about undeleted dynamically created objects at the unload of a mql5 program

See also

[Data Types](#), [Encapsulation and Extensibility of Types](#), [Initialization of Variables](#), [Visibility Scope and Lifetime of Variables](#)

Preprocessor

Preprocessor is a special subsystem of the MQL5 compiler that is intended for preparation of the program source code immediately before the program is compiled.

Preprocessor allows enhancement of the source code readability. The code can be structured by including of specific files containing source codes of mql5-programs. The possibility to assign mnemonic names to specific constants contributes to enhancement of the code readability.

Preprocessor also allows determining specific parameters of mql5-programs:

- [Declare constants](#)
- [Set program properties](#)
- [Include files in program text](#)
- [Import functions](#)
- [Conditional Compilation](#)

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

Macro substitution (#define)

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

```
#define identifier expression           // parameter-free form
#define identifier(par1,... par8) expression // parametric form
```

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

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

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

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

Example:

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

Parametric Form #define

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

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



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

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

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

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

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

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

// incorrect parametric form
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9) p1+p2+p3+p4 // more than 8 parameters
```

The #undef directive

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

Example:

```
#define MACRO

void func1()
{
    #ifdef MACRO
        Print("MACRO is defined in ", __FUNCTION__);
    #else
```



```
    Print("MACRO is not defined in ", __FUNCTION__);
#endif
}

#undef MACRO

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

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

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

See also

[Identifiers](#), [Character Constants](#)

Program Properties (#property)

Every mql5-program allows to specify additional specific parameters named `#property` that help client terminal in proper servicing for programs without the necessity to launch them explicitly. This concerns external settings of indicators, first of all. Properties described in included files are completely ignored. Properties must be specified in the main mq5-file.

```
#property identifier value
```

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

Constant	Type	Description
icon	string	Path to the file of an image that will be used as an icon of the EX5 program. Path specification rules are the same as for resources . The property must be specified in the main module with the MQL5 source code. The icon file must be in the ICO format.
link	string	Link to the company website
copyright	string	The company name
version	string	Program version, maximum 31 characters
description	string	Brief text description of a mql5-program. Several <i>description</i> can be present, each of them describes one line of the text. The total length of all <i>description</i> can not exceed 511 characters including line feed.
stacksize	int	MQL5 program stack size. The stack of sufficient size is necessary when executing function recursive calls. When launching a script or an Expert Advisor on the chart, the stack of at least 8 MB is allocated. In case of indicators, the stack size is always fixed and equal to 1 MB. When a program is launched in the strategy tester, the stack

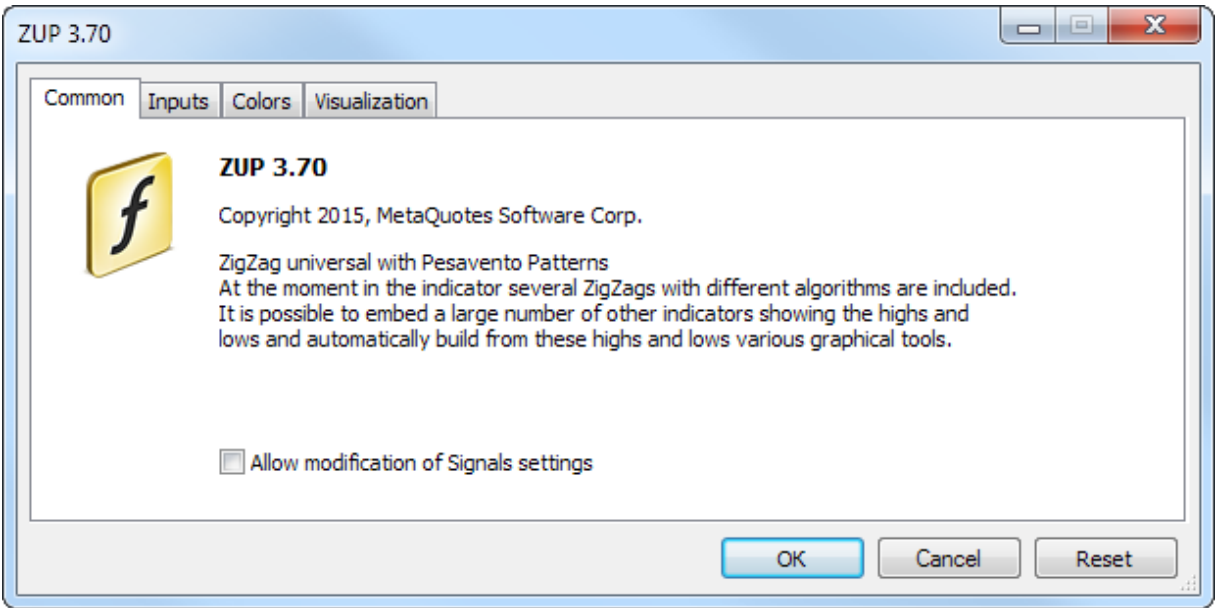
		of 16 MB is always allocated for it.
library		A library; no start function is assigned, functions with the export modifier can be imported in other mql5-programs
indicator_applied_price	int	Specifies the default value for the "Apply to" field. You can specify one of the values of ENUM_APPLIED_PRICE . If the property is not specified, the default value is PRICE_CLOSE
indicator_chart_window		Show the indicator in the chart window
indicator_separate_window		Show the indicator in a separate window
indicator_height	int	Fixed height of the indicator subwindow in pixels (property INDICATOR_HEIGHT)
indicator_buffers	int	Number of buffers for indicator calculation
indicator_plots	int	Number of graphic series in the indicator
indicator_minimum	double	The bottom scaling limit for a separate indicator window
indicator_maximum	double	The top scaling limit for a separate indicator window
indicator_labelN	string	Sets a label for the N-th graphic series displayed in DataWindow. For graphic series requiring multiple indicator buffers (DRAW_CANDLES, DRAW_FILLING and others), the label names are defined using the separator ';'.
indicator_colorN	color	The color for displaying line N, where N is the number of graphic series ; numbering starts from 1
indicator_widthN	int	Line thickness in graphic series , where N is the number

		of graphic series; numbering starts from 1
indicator_styleN	<u>int</u>	Line style in <u>graphic series</u> , specified by the values of <u>ENUM_LINE_STYLE</u> . N is the number of graphic series; numbering starts from 1
indicator_typeN	<u>int</u>	Type of graphical plotting, specified by the values of <u>ENUM_DRAW_TYPE</u> . N is the number of graphic series; numbering starts from 1
indicator_levelN	<u>double</u>	Horizontal level of N in a separate indicator window
indicator_levelcolor	<u>color</u>	Color of horizontal levels of the indicator
indicator_levelwidth	<u>int</u>	Thickness of horizontal levels of the indicator
indicator_levelstyle	<u>int</u>	Style of horizontal levels of the indicator
script_show_confirm		Display a confirmation window before running the script
script_show_inputs		Display a window with the properties before running the script and disable this confirmation window
tester_indicator	<u>string</u>	Name of a custom indicator in the format of "indicator_name.ex5". Indicators that require testing are defined automatically from the call of the <u>iCustom()</u> function, if the corresponding parameter is set through a constant string. For all other cases (use of the <u>IndicatorCreate()</u> function or use of a non-constant string in the parameter that sets the indicator name) this property is required
tester_file	<u>string</u>	File name for a tester with the indication of extension, in double quotes (as a constant

		string). The specified file will be passed to tester. Input files to be tested, if there are necessary ones, must always be specified.
tester_library	<u>string</u>	Library name with the extension, in double quotes. A library can have 'dll' or 'ex5' as file extension. Libraries that require testing are defined automatically. However, if any of libraries is used by a <u>custom</u> indicator, this property is required

Sample Task of Description and Version Number

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



Examples of Specifying a Separate Label for Each Indicator Buffer ("C open; C high; C low; C close")

```
#property indicator_chart_window
#property indicator_buffers 4
#property indicator_plots   1
#property indicator_type1   DRAW_CANDLES
#property indicator_width1  3
```



```
#property indicator_label1 "C open;C high;C low;C close"
```



Including Files (#include)

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

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

Examples:

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

The preprocessor replaces the line *#include <file_name>* with the content of the file WinUser32.mqh. Angle brackets indicate that the WinUser32.mqh file will be taken from the standard directory (usually it is *terminal_installation_directory\MQL5\Include*). The current directory is not included in the search.

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

See also

[Standard Library](#), [Importing Functions](#)

Importing Function (#import)

Functions are imported from compiled MQL5 modules (*.ex5 files) and from operating system modules (*.dll files). The module name is specified in the `#import` directive. For compiler to be able to correctly form the imported function call and organize proper [transmission parameters](#), the full description of [functions](#) is needed. Function descriptions immediately follow the `#import "module name"` directive. New command `#import` (can be without parameters) completes the block of imported function descriptions.

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

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

The order of searching for a file specified after the `#import` keyword is described in [Call of Imported Functions](#).

Since the imported functions are outside the compiled module, the compiler can not verify the validity of passed parameters. Therefore, to avoid run-time errors, one must accurately describe the composition and order of parameters passed to imported functions. Parameters passed to imported functions (both from EX5, and from the DLL-module) can have default values.

The following can't be used for parameters in imported functions:

- [pointers](#) (*);
- links to objects that contain [dynamic arrays](#) and/or pointers.

Classes, string arrays or complex objects that contain strings and/or dynamic arrays of any types cannot be passed as a parameter to functions imported from DLL.

Examples:

```
#import "user32.dll"
int    MessageBoxW(uint hWnd,string lpText,string lpCaption,uint uType);
#import "stdlib.ex5"
string ErrorDescription(int error_code);
int    RGB(int red_value,int green_value,int blue_value);
bool   CompareDoubles(double number1,double number2);
string DoubleToStrMorePrecision(double number,int precision);
string IntegerToHexString(int integer_number);
#import "ExpertSample.dll"
int    GetIntValue(int);
double GetDoubleValue(double);
string GetStringValue(string);
double GetArrayItemValue(double &arr[],int,int);
bool   SetArrayItemValue(double &arr[],int,int,double);
```



```
double GetRatesItemValue(double &rates[][6],int,int,int);  
#import
```

To import functions during execution of a mql5 program, early binding is used. This means that the library is loaded during the loading of a program using its ex5 program.

It's not recommended to use a fully qualified name of the loadable module of type *Drive:\Directory\FileName.Ext*. MQL5 libraries are loaded from the *terminal_dir\MQL5\Libraries* folder.

See also

[Including Files](#)

Conditional Compilation (#ifdef, #ifndef, #else, #endif)

Preprocessor conditional compilation directives allow compiling or skipping a part of the program depending on the fulfillment of a certain condition.

That condition can take one of the following forms.

```
#ifdef identifier
    // the code located here is compiled if the identifier has already been defined for
#endif
```

```
#ifndef identifier
    // the code located here is compiled if the identifier is not currently defined by
#endif
```

Any of the conditional compilation directives can be followed by any number of lines possibly containing #else directive and ending with #endif. If the verified condition is true, the lines between #else and #endif are ignored. If the verified condition is not fulfilled, all lines between checking and #else directive (or #endif directive if the former is absent) are ignored.

Example:

```
#ifndef TestMode
    #define TestMode
#endif
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    #ifdef TestMode
        Print("Test mode");
    #else
        Print("Normal mode");
    #endif
}
```

Depending on the program type and compilation mode, the standard macros are defined the following way:

__MQL5__ macro is defined when compiling *.mq5 file, __MQL4__ macro is defined when compiling *.mq4 one.

_DEBUG macro is defined when compiling in debug mode.

_RELEASE macro is defined when compiling in release mode.

Example:

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



```
#ifdef __MQL5__
    #ifdef _DEBUG
        Print("Hello from MQL5 compiler [DEBUG]");
    #else
        #ifdef _RELEASE
            Print("Hello from MQL5 compiler [RELEASE]");
        #endif
    #endif
#else
    #ifdef __MQL4__
        #ifdef _DEBUG
            Print("Hello from MQL4 compiler [DEBUG]");
        #else
            #ifdef _RELEASE
                Print("Hello from MQL4 compiler [RELEASE]");
            #endif
        #endif
    #endif
#endif
#endif
}
```


Object-Oriented Programming

Object-oriented programming (OOP) is programming primarily focused on data, while data and behavior are being inseparably linked. Data and behavior together constitute a class, while objects are class instances.

The components of the object-oriented approach are:

- [Encapsulation and type extensibility](#)
- [Inheritance](#)
- [Polymorphism](#)
- [Overloading](#)
- [Virtual functions](#)

OOP considers computation as modeling of behavior. The modeled item is the object represented by computational abstractions. Suppose we want to write a well known game "Tetris". To do this, we must learn how to model the appearance of random shapes composed of four squares joined together by edges. Also we need to regulate the falling speed of shapes, define operations of rotation and shift of shapes. Moving of shapes on the screen is limited by the well's boundaries, this requirement must also be modeled. Besides that, filled rows of cubes must be destroyed and achieved points must be counted.

Thus, this easy-to-understand game requires the creation of several models - shape model, well model, shape movement model and so on. All these models are abstractions, represented by calculations in the computer. To describe these models, the concept of Abstract Data Type, ADT (or [complex data type](#)) is used. Strictly speaking, the model of the "shapes" motion in the DOM is not a data type, but it is a set of operations on the "shape" data type, using the restrictions of the "well" data type.

Objects are [class](#) variables. Object-oriented programming allows you to easily create and use ADT. Object-oriented programming uses the inheritance mechanism. The benefit of inheritance is in the fact that it allows obtaining derivative types from data types already defined by a user.

For example, to create Tetris shapes, it's convenient to create a base class Shape first. The other classes representing all seven possible shape types can be derived on its basis. Behavior of shapes is defined in the base class, while implementation of behavior of each separate shape is defined in derivative classes.

In OOP objects are responsible for their behavior. ADT developer should include a code to describe any behavior that would normally be expected from the corresponding objects. The fact that the object itself is responsible for its behavior, greatly simplifies the task of programming for the user of this object.

If we want to draw a shape on the screen, we need to know where the center will be and how to draw it. If a separate shape knows how to draw itself, the programmer should send a "draw" message when using such a shape.

The MQL5 Language is a C++ like, and it also has the [encapsulation](#) mechanism for the implementation of ADT. On the one hand encapsulation combines the internal details of the implementation of a particular type, and on the other hand it combines externally accessible functions that can influence objects of this type. Implementation details may be inaccessible for a program that uses this type.

The concept of OOP has a set of related concepts, including the following:

- Simulation of actions from the real world
- User-defined data types
- Hiding the implementation details
- Possibility of the code reuse through inheritance
- Interpretation of function calls during execution

Some of these concepts are rather vague, some are abstract, others are general.

Encapsulation and Extensibility of Types

OOP is a balanced approach to writing software. Data and behavior are packed together. This encapsulation creates user-defined data types, extending the language data types and interacting with them. Types extensibility is an opportunity to add to the language user-defined data types, which are also easy to use, as well as [basic types](#).

An abstract data type, for example, a string, is a description of the ideal, well known behavior type.

The string user knows that the string operations, such as concatenation or print, have a certain behavior. Concatenation and print operations are called methods.

A certain implementation of ADT may have some restrictions, for example, strings can be limited in length. These limitations affect the behavior opened to all. At the same time, internal or private implementation details do not affect directly the way the user sees the object. For example, the string is often implemented as an array, while the internal base address of this array and its name are not essential for the user.

Encapsulation is the ability to hide the implementation details when the open interfaces to user-defined type is provided. In MQL5, as well as in C++, class and structure definitions ([class](#) and [struct](#)) are used for the encapsulation provisions in combination with access keywords [private](#), [protected](#) and [public](#).

The [public](#) keyword shows that access to the members that stand behind it is open without restrictions. Without this keyword, class members are locked by default. Private members are accessible only by member functions only of its class.

Protected class functions are available to class functions not only in its class, but also in its inheritor classes. Public class functions are available for any function within the scope of the class declaration. The protection makes possible to hide part of the class implementation, thus preventing unexpected changes in the structure of data. Access restriction or data hiding is a feature of the object-oriented programming.

Usually, class functions are protected and declared with the [protected](#) modifier, the reading and writing of the values are performed by using special so-called set-and get-methods that are defined by the [public](#) access modifier.

Example:

```
class CPerson
{
protected:
    string          m_name;           // name
public:
    void            SetName(string n) {m_name=n;} // sets name
    string          GetName() {return (m_name);} // returns name
};
```


This approach offers several advantages. First, by function name we can understand what it does - sets or gets the value of a class member. Secondly, perhaps in the future we will need to change the type of the m_name variable in the CPerson class or in any of its derivative classes.

In this case, we'll need just to change the implementation of functions SetName() and GetName(), while objects of the CPerson class will be available for using in a program without any code changes because the user will not even know that the data type of m_name has changed.

Example:

```
struct Name
{
    string      first_name;           // name
    string      last_name;           // last name
};

class CPerson
{
protected:
    Name        m_name;               // name
public:
    void        SetName(string n);
    string      GetName() {return(m_name.first_name+" "+m_name.last_name);}
private:
    string      GetFirstName(string full_name);
    string      GetLastName(string full_name);
};

void CPerson::SetName(string n)
{
    m_name.first_name=GetFirstName(n);
    m_name.last_name=GetLastName(n);
}

string CPerson::GetFirstName(string full_name)
{
    int pos=StringFind(full_name," ");
    if(pos>0) StringSetCharacter(full_name,pos,0);
    return(full_name);
}

string CPerson::GetLastName(string full_name)
{
    string ret_string;
    int pos=StringFind(full_name," ");
    if(pos>0) ret_string=StringSubstr(full_name,pos+1);
    else      ret_string=full_name;
    return(ret_string);
}
```


See also

[Data Types](#)

Inheritance

The characteristic feature of OOP is the encouragement of code reuse through inheritance. A new class is made from the existing, which is called the base class. The derived class uses the members of the base class, but can also modify and supplement them.

Many types are variations of the existing types. It is often tedious to develop a new code for each of them. In addition, the new code implies new errors. The derived class inherits the description of the base class, thus any re-development and re-testing of code is unnecessary. The inheritance relationships are hierarchical.

Hierarchy is a method that allows to copy the elements in all their diversity and complexity. It introduces the objects classification. For example, the periodic table of elements has gases. They possess to properties inherent to all periodic elements.

Inert gases constitute the next important subclass. The hierarchy is that the inert gas, such as argon is a gas, and gas, in its turn, is part of the system. Such a hierarchy allows to interpret behaviour of inert gases easily. We know that their atoms contain protons and electrons, that is true for all other elements.

We know that they are in a gaseous state at room temperature, like all the gases. We know that no gas from inert gas subclass enters usual chemical reaction with other elements, and it is a property of all inert gases.

Consider an example of the inheritance of geometric shapes. To describe the whole variety of simple shapes (circle, triangle, rectangle, square etc.), the best way is to create a base class ([ADT](#)), which is the ancestor of all the derived classes.

Let's create a base class CShape, which contains just the most common members describing the shape. These members describe properties that are characteristic of any shape - the type of the shape and main anchor point coordinates.

Example:

```
//--- The base class Shape
class CShape
{
protected:
    int      m_type;           // Shape type
    int      m_xpos;           // X - coordinate of the base point
    int      m_ypos;           // Y - coordinate of the base point
public:
    CShape() {m_type=0; m_xpos=0; m_ypos=0;} // constructor
    void     SetXPos(int x) {m_xpos=x;} // set X
    void     SetYPos(int y) {m_ypos=y;} // set Y
};
```

Next, create new classes derived from the base class, in which we will add necessary fields, each specifying a certain class. For the Circle shape it is necessary to add a member that contains the radius value. The Square shape is characterized by the side value. Therefore, derived classes, inherited from the base class CShape will be declared as follows:


```
//--- The derived class circle
class CCircle : public CShape      // After a colon we define the base class
{                                  // from which inheritance is made
private:
    int          m_radius;        // circle radius

public:
    CCircle() {m_type=1;} // constructor, type 1
};
```

For the Square shape class declaration is similar:

```
//--- the derived class Square
class CSquare : public CShape      // After a colon we define the base class
{                                  // from which inheritance is made
private:
    int          m_square_side;    // square side

public:
    CSquare() {m_type=2;} // constructor, type 2
};
```

It should be noted that while object is created the base class constructor is called first, and then the [constructor](#) of the derived class is called. When an object is destroyed first the [destructor](#) of the derived class is called, and then a base class destructor is called.

Thus, by declaring the most general members in the base class, we can add an additional members in derived classes, which specify a particular class. Inheritance allows creating powerful code libraries that can be reused many times.

The syntax for creating a derived class from an already existing one is as follows:

```
class class_name :
    (public | protected | private) opt base_class_name
{
    class members declaration
};
```

One of aspects of the derived class is the visibility (openness) of its members successors (heirs). The public, protected and private keywords are used to indicate the extent, to which members of the base class will be available for the derived one. The public keyword after a colon in the header of a derived class indicates that the protected and public members of the base class CShape should be inherited as protected and public members of the derived class CCircle.

The private class members of the base class are not available for the derived class. The public inheritance also means that derived classes (CCircle and CSquare) are CShapes. That is, the Square (CSquare) is a shape (CShape), but the shape does not necessarily have to be a square.

The derived class is a modification of the base class, it inherits the protected and public members of the base class. The constructors and destructors of the base class cannot be inherited. In addition to members of the base class, new members are added in a derivative class.

The derived class may include the implementation of member functions, different from the base class. It has nothing common with an [overload](#), when the meaning of the same function name may be different for different signatures.

In protected inheritance, public and protected members of base class become protected members of derived class. In private inheritance, the public and protected members of base class become private members of the derived class.

In protected and private inheritance, the relation that "the object of a derivative class is object of a base class" is not true. The protected and private inheritance types are rare, and each of them needs to be used carefully.

It should be understood that the type of inheritance (public, protected or private) does not affect the ways of **accessing the members of base classes in the hierarchy of inheritance from a derived class**. With any type of inheritance, only base class members declared with public and protected access specifiers will be available out of the derived classes. Let's consider it in the following example:

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+-----+
//| Example class with a few access types |
//+-----+

class CBaseClass
{
private:          //--- The private member is not available from derived classes
    int          m_member;
protected:      //--- The protected method is available from the base class and
    int          Member() {return(m_member); }
public:          //--- Class constructor is available to all members of classes
    CBaseClass() {m_member=5;return;};
private:        //--- A private method for assigning a value to m_member
    void         Member(int value) { m_member=value;};

};

//+-----+
//| Derived class with errors |
//+-----+

class CDerived: public CBaseClass // specification of public inheritance can be omitted
{
public:
    void Func() // In the derived class, define a function with calls to base class members
    {
        //--- An attempt to modify a private member of the base class
        m_member=0;          // Error, the private member of the base class is not available
        Member(0);           // Error, the private method of the base class is not available
    }
};
```



```

    //--- Reading the member of the base class
    Print(m_member);    // Error, the private member of the base class is not available
    Print(Member());    // No error, protected method is available from the base class
}
};

```

In the above example, CBaseClass has only a public method - the constructor. Constructors are called automatically when creating a class object. Therefore, the private member m_member and the protected methods Member() cannot be called from the outside. But in case of public inheritance, the Member() method of the base class will be available from the derived classes.

In case of protected inheritance, all the members of the base class with public and protected access become protected. It means that if public data members and methods of the base class were accessible from the outside, with protected inheritance they are available only from the classes of the derived class and its further derivatives.

```

//+-----+
//| Example class with a few access types |
//+-----+
class CBaseMathClass
{
private:    //--- The private member is not available from derived classes
    double    m_Pi;
public:    //--- Getting and setting a value for m_Pi
    void    SetPI(double v){m_Pi=v;return;};
    double    GetPI(){return m_Pi;};
public:    // The class constructor is available to all members
    CBaseMathClass() {SetPI(3.14); PrintFormat("%s",__FUNCTION__);};
};

//+-----+
//| Derived class, in which m_Pi cannot be modified |
//+-----+
class CProtectedChildClass: protected CBaseMathClass // Protected inheritance
{
private:
    double    m_radius;
public:    //--- Public methods in the derived class
    void    SetRadius(double r){m_radius=r; return;};
    double    GetCircleLength(){return GetPI()*m_radius;};
};

//+-----+
//| Script starting function |
//+-----+
void OnStart()
{
    //--- When creating a derived class, the constructor of the base class will be called
    CProtectedChildClass pt;
    //--- Specify radius
    pt.SetRadius(10);
    PrintFormat("Length=%G",pt.GetCircleLength());
}

```



```
//--- If we uncomment the line below, we will get an error at the stage of compilation
// pt.SetPI(3);

//--- Now declare a variable of the base class and try to set the Pi constant equal to
    CBaseMathClass bc;
    bc.SetPI(10);
//--- Here is the result
    PrintFormat("bc.GetPI()=%G",bc.GetPI());
}
```

The example shows that methods SetPI() and GetPI() in the base class CBaseMathClass are open and available for calling from any place of the program. But at the same time, for CProtectedChildClass which is derived from it these methods can be called only from the methods of the CProtectedChildClass class or its derived classes.

In case of private inheritance, all the members of the basic class with the public and protected access become private, and calling them becomes impossible in further inheritance.

MQL5 has no multiple inheritance.

See also

[Structures and Classes](#)

Polymorphism

Polymorphism is an opportunity for different classes of objects, related through inheritance, to respond in various ways when calling the same function element. It helps to create a universal mechanism describing the behavior of not only the base class, but also descendant classes.

Let's continue to develop a base class CShape, and define a member function GetArea(), designed to calculate the area of a shape. In all the descendant classes, produced by inheritance from the base class, we redefine this function in accordance with rules of calculating the area of a particular shape.

For a square (class CSquare), the area is calculated through its sides, for a circle (class CCircle), area is expressed through its radius etc. We can create an array to store objects of CShape type, in which both objects of a base class and those of all descendant classes can be stored. Further we can call the same function for each element of the array.

Example:

```
//--- Base class
class CShape
{
protected:
    int          m_type;           // Shape type
    int          m_xpos;           // X - coordinate of the base point
    int          m_ypos;           // Y - coordinate of the base point
public:
    void         CShape() {m_type=0;}; // constructor, type=0
    int          GetType() {return (m_type);}; // returns type of the shape
    virtual
    double       GetArea() {return (0);}; // returns area of the shape
};
```

Now, all of the derived classes have a member function getArea(), which returns a zero value. The implementation of this function in each descendant will vary.

```
//--- The derived class Circle
class CCircle : public CShape           // After a colon we define the base class
{                                       // from which inheritance is made
private:
    double       m_radius;           // circle radius

public:
    void         CCircle() {m_type=1;}; // constructor, type=1
    void         SetRadius(double r) {m_radius=r;};
    virtual double GetArea() {return (3.14*m_radius*m_radius);}; // circle area
};
```

For the class Square the declaration is the same:

```
//--- The derived class Square
class CSquare : public CShape           // After a colon we define the base class
{                                       // from which inheritance is made
```



```

private:
    double          m_square_side;          // square side

public:
    void            CSquare(){m_type=2;}; // constructor, type=1
    void            SetSide(double s){m_square_side=s;};
    virtual double  GetArea(){return (m_square_side*m_square_side);} // square area
};

```

For calculating the area of the square and circle, we need the corresponding values of `m_radius` and `m_square_side`, so we have added the functions `SetRadius()` and `SetSide()` in the declaration of the corresponding class.

It is assumed that object of different types (`CCircle` and `CSquare`) derived from one base type `CShape` are used in our program. Polymorphism allows creating an array of objects of the base `CShape` class, but when declaring this array, these objects are yet unknown and their type is undefined.

The decision on what type of object will be contained in each element of the array will be taken directly during program execution. This involves the [dynamic creation](#) of objects of the appropriate classes, and hence the necessity to use [object pointers](#) instead of objects.

The [new](#) operator is used for dynamic creation of objects. Each such object must be individually and explicitly deleted using the [delete](#) operator. Therefore we will declare an array of pointers of `CShape` type, and create an object of a proper type for each element (**new Class_Name**), as shown in the following script example:

```

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- Declare an array of object pointers of the base type
    CShape *shapes[5]; // An array of pointers to CShape object

    //--- Here fill in the array with derived objects
    //--- Declare a pointer to the object of CCircle type
    CCircle *circle=new CCircle();
    //--- Set object properties at the circle pointer
    circle.SetRadius(2.5);
    //--- Place the pointer value in shapes[0]
    shapes[0]=circle;

    //--- Create another CCircle object and write down its pointer in shapes[1]
    circle=new CCircle();
    shapes[1]=circle;
    circle.SetRadius(5);

    //--- Here we intentionally "forget" to set a value for shapes[2]
    //circle=new CCircle();
    //circle.SetRadius(10);
    //shapes[2]=circle;

```



```

//--- Set NULL for the element that is not used
shapes[2]=NULL;

//--- Create a CSquare object and write down its pointer to shapes[3]
CSquare *square=new CSquare();
square.SetSide(5);
shapes[3]=square;

//--- Create a CSquare object and write down its pointer to shapes[4]
square=new CSquare();
square.SetSide(10);
shapes[4]=square;

//--- We have an array of pointers, get its size
int total=ArraySize(shapes);
//--- Pass in a loop through all pointers in the array
for(int i=0; i<5;i++)
{
    //--- If the pointer at the specified index is valid
    if(CheckPointer(shapes[i])!=POINTER_INVALID)
    {
        //--- Log the type and square of the shape
        PrintFormat("The object of type %d has the square %G",
            shapes[i].GetType(),
            shapes[i].GetArea());
    }
    //--- If the pointer has type POINTER_INVALID
    else
    {
        //--- Notify of an error
        PrintFormat("Object shapes[%d] has not been initialized! Its pointer is %s",
            i,EnumToString(CheckPointer(shapes[i])));
    }
}

//--- We must delete all created dynamic objects
for(int i=0;i<total;i++)
{
    //--- We can delete only the objects with pointers of POINTER_DYNAMIC type
    if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
    {
        //--- Notify of deletion
        PrintFormat("Deleting shapes[%d]",i);
        //--- Delete an object by its pointer
        delete shapes[i];
    }
}
}

```


Please note that when deleting an object using the [delete](#) operator, [the type of its pointer](#) must be checked. Only objects with the [POINTER_DYNAMIC](#) pointer can be deleted using delete. For pointers of other type, an error will be returned.

But besides the redefining of functions during inheritance, polymorphism also includes the implementation of one and the same functions with different sets of parameters within a class. This means that the class may have several functions with the same name but with a different type and/or set of parameters. In this case, polymorphism is implemented through the [function overload](#).

See also

[Standard Library](#)

Overload

Within one class it is possible to define two or more methods that use the same name, but have different numbers of parameters. When this occurs, methods are called overloaded and such a process is referred to as method overloading.

Method overloading is one of ways of [polymorphism](#) realization. Overloading of methods is performed according to the same rules as the [function overloading](#).

If the called function has no exact match, the compiler searches for a suitable function on three levels sequentially:

1. search within class methods.
2. search within the base class methods, consistently from the nearest ancestor to the very first.
3. search among other functions.

If there is no exact correspondence at all levels, but several suitable functions at different levels have been found, the function found at the least level is used. Within one level, there can't be more than one suitable function.

See also

[Function Overloading](#)

Virtual Functions

The virtual keyword is the function specifier, which provides a mechanism to select dynamically at runtime an appropriate function-member among the functions of basic and derived classes. Structures cannot have virtual functions. It can be used to change the [declarations](#) for function-members only.

The virtual function, like an ordinary function, must have an [executable body](#). When called, its semantic is the same as that of other functions.

A virtual function may be overridden in a derived class. The choice of what [function definition](#) should be called for a virtual function is made dynamically (at runtime). A typical case is when a base class contains a virtual function, and derived classes have their own versions of this function.

The pointer to the base class can indicate either a base class object or the object of a derived class. The choice of the member-function to call will be performed at runtime and will depend on the type of the object, not the type of the pointer. If there is no member of a derived type, the virtual function of the base class is used by default.

[Destructors](#) are always virtual, regardless of whether they are declared with the [virtual](#) keyword or not.

Let's consider the use of virtual functions on the example of MT5_Tetris.mq5. The base class CTetrisShape with the virtual function Draw is defined in the included file MT5_TetisShape.mqh.

```
//+-----+
class CTetrisShape
{
protected:
    int          m_type;
    int          m_xpos;
    int          m_ypos;
    int          m_xsize;
    int          m_ysize;
    int          m_prev_turn;
    int          m_turn;
    int          m_right_border;
public:
    void          CTetrisShape();
    void          SetRightBorder(int border) { m_right_border=border; }
    void          SetYPos(int ypos)         { m_ypos=ypos;           }
    void          SetXPos(int xpos)         { m_xpos=xpos;           }
    int           GetYPos()                  { return(m_ypos);         }
    int           GetXPos()                  { return(m_xpos);         }
    int           GetYSize()                 { return(m_ysize);        }
    int           GetXSize()                 { return(m_xsize);        }
    int           GetType()                  { return(m_type);         }
    void          Left()                    { m_xpos-=SHAPE_SIZE;      }
    void          Right()                   { m_xpos+=SHAPE_SIZE;      }
    void          Rotate()                   { m_prev_turn=m_turn; if(++m_turn>3) m_turn=0; }
    virtual void  Draw()                    { return;                  }
    virtual bool  CheckDown(int& pad_array[]);
    virtual bool  CheckLeft(int& side_row[]);
}
```



```
virtual bool      CheckRight(int& side_row[]);
};
```

Further, for each derived class, this function is implemented in accordance with characteristics of a descendant class. For example, the first shape CTetrisShape1 has its own implementation of the Draw() function:

```
class CTetrisShape1 : public CTetrisShape
{
public:
    ///--- shape drawing
    virtual void      Draw()
    {
        int      i;
        string name;
        ///---
        if(m_turn==0 || m_turn==2)
        {
            ///--- horizontal
            for(i=0; i<4; i++)
            {
                name=SHAPE_NAME+(string)i;
                ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
                ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
            }
        }
        else
        {
            ///--- vertical
            for(i=0; i<4; i++)
            {
                name=SHAPE_NAME+(string)i;
                ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos);
                ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+i*SHAPE_SIZE);
            }
        }
    }
};
```

The Square shape is described by class CTetrisShape6 and has its own implementation of the Draw() method:

```
class CTetrisShape6 : public CTetrisShape
{
public:
    ///--- Shape drawing
    virtual void      Draw()
    {
        int      i;
        string name;
```



```

//---
for(i=0; i<2; i++)
{
    name=SHAPE_NAME+(string)i;
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
}
for(i=2; i<4; i++)
{
    name=SHAPE_NAME+(string)i;
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+(i-2)*SHAPE_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+SHAPE_SIZE);
}
}
};

```

Depending on the class, to which the created object belongs, it calls the virtual function of this or that derived class.

```

void CTetrisField::NewShape()
{
    //--- creating one of the 7 possible shapes randomly
    int nshape=rand()%7;
    switch(nshape)
    {
        case 0: m_shape=new CTetrisShape1; break;
        case 1: m_shape=new CTetrisShape2; break;
        case 2: m_shape=new CTetrisShape3; break;
        case 3: m_shape=new CTetrisShape4; break;
        case 4: m_shape=new CTetrisShape5; break;
        case 5: m_shape=new CTetrisShape6; break;
        case 6: m_shape=new CTetrisShape7; break;
    }
    //--- draw
    m_shape.Draw();
    //---
}

```

Modifier 'override'

The 'override' modifier means that the declared function must override the method of a parent class. Use of this method allows you to avoid overriding errors, for example it allows you to avoid accidental modification of the method signature. Suppose, the 'func' method is defined in the base class. The method accepts an int variable as an argument:

```

class CFoo
{
    void virtual func(int x) const { }
};

```

Next, the method is overridden in the child class:


```
class CBar : public CFoo
{
    void func(short x) { }
};
```

However, the argument type is mistakenly changed from int to short. In fact, this is not method overriding, but it is method overloading. Acting in accordance with the [overloaded function defining algorithm](#), the compiler can in certain situations choose a method defined in the base class instead of the overridden method.

In order to avoid such errors, you should explicitly add the 'override' modifier to the method you want to override.

```
class CBar : public CFoo
{
    void func(short x) override { }
};
```

If the method signature is changed during overriding, the compiler will not be able to find a method with the same signature in the parent class, and it will return a compilation error:

```
'CBar::func' method is declared with 'override' specifier but does not override any base
```

Modifier 'final'

The 'final' modifier does the opposite – it prohibits method overriding in child classes. If a method implementation is sufficient and fully complete, declare this method with the 'final' modifier so as to make sure that it will not be modified later.

```
class CFoo
{
    void virtual func(int x) final { }
};

class CBar : public CFoo
{
    void func(int) { }
};
```

If you try to override a method with the 'final' modifier as shown in the above example, the compiler will return an error:

```
'CFoo::func' method declared as 'final' cannot be overridden by 'CBar::func'
see declaration of 'CFoo::func'
```

See also

[Standard Library](#)

Static members of a Class/Structure

Static Members

The members of a class can be declared using the storage class modifier [static](#). These data members are shared by all instances of this class and are stored in one place. Non-static data members are created for each class object variable.

The inability to declare static members of a class would have led to the need to declare these data on the [the global level](#) of the program. It would break the relationship between the data and their class, and is not consistent with the basic paradigm of the OOP - joining data and methods for handling them in a class. The static member allows class data that are not specific to a particular instance to exist in the class scope.

Since a static class member does not depend on the particular instance, the reference to it is as follows:

```
class_name::variable
```

where *class_name* is the name of the class, and *variable* is the name of the class member.

As you see, to access the static member of a class, [context resolution operator ::](#) is used. When you access a static member within class methods, the context operator is optional.

Static member of a class has to be explicitly initialized with desired value. For this it must be declared and initialized in global scope. The sequence of static members initialization will correspond to the sequence of their declaration in global scope.

For example, we have a class *CParser* used for parsing the text, and we need to count the total number of processed words and characters. We only need to declare the necessary class members as static and initialize them at the global level. Then all instances of the class will use common counters of words and characters.

```
//+-----+
//| Class "Text analyzer" |
//+-----+
class CParser
{
public:
    static int      s_words;
    static int      s_symbols;
    //-- Constructor and destructor
                   CParser(void);
                   ~CParser(void) {};
};
...
/-- Initialization of static members of the Parser class at the global level
int CParser::s_words=0;
int CParser::s_symbols=0;
```

A static class member can be declared with the *const* keyword. Such static constants must be initialized at the global level with the *const* keyword:


```
//+-----+
//| Class "Stack" for storing processed data |
//+-----+
class CStack
{
public:
    CStack(void);
    ~CStack(void){};

...
private:
    static const int s_max_length; // Maximum stack capacity
};

//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
```

Pointer this

The keyword [this](#) denotes an implicitly declared [pointer](#) to itself - to a specific instance of the class, in the context of which the method is executed. It can be used only in non-static methods of the class. Pointer `this` is an implicit non-static member of any class.

In static functions you can access only static members/methods of a class.

Static Methods

In MQL5 member functions of type [static](#) can be used. The *static* modifier must precede the return type of a function in the declaration inside a class.

```
class CStack
{
public:
    //--- Constructor and destructor
    CStack(void){};
    ~CStack(void){};

    //--- Maximum stack capacity
    static int Capacity();
private:
    int m_length; // The number of elements in the stack
    static const int s_max_length; // Maximum stack capacity
};

//+-----+
//| Returns the maximum number of elements to store in the stack |
//+-----+
int CStack::Capacity(void)
{
    return(s_max_length);
}

//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
```



```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- declare CStack type variable
    CStack stack;
//--- call the object's static method
    Print("CStack.s_max_length=",stack.Capacity());
//--- it can also be called the following way, as the method is static and does not re
    Print("CStack.s_max_length=",CStack::Capacity());
}
```

A method with the **const** modifier is called constant and cannot modify implicit members of its class. Declaration of constant functions of a class and constant parameters is called *const-correctness* control. Through this control you can be sure that the compiler will ensure the consistency of values of objects and will return an error during compilation if there is something wrong.

The **const** modifier is placed after the list of arguments inside a class declaration. Definition outside a class should also include the *const* modifier:

```
//+-----+
//| Class "Rectangle" |
//+-----+
class CRectangle
{
private:
    double      m_width;      // Width
    double      m_height;     // Height
public:
    //--- Constructors and destructor
        CRectangle(void):m_width(0),m_height(0){};
        CRectangle(const double w,const double h):m_width(w),m_height(h){};
        ~CRectangle(void){};

    //--- Calculating the area
    double      Square(void) const;
    static double Square(const double w,const double h); // { return(w*h); }
};

//+-----+
//| Returns the area of the "Rectangle" object |
//+-----+
double CRectangle::Square(void) const
{
    return(Square(m_width,m_height));
}

//+-----+
//| Returns the product of two variables |
//+-----+
static double CRectangle::Square(const double w,const double h)
{
}
```



```
        return(w*h);
    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //-- Create a rectangle rect with the sides equal to 5 and 6
    CRectangle rect(5,6);
    //-- Find the rectangle area using a constant method
    PrintFormat("rect.Square()=%.2f",rect.Square());
    //-- Find the product of numbers using the static method of class CRectangle
    PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5));
}
```

An additional argument in favor of using the constancy control is the fact that in this case, the compiler generates a special optimization, for example, places a constant object in read-only memory.

A static function cannot be determined with the `const` modifier, because this modifier ensures the constancy of the instance members when calling this function. But, as mentioned above, the static function cannot access non-static class members.

See also

[Static Variables](#), [Variables](#), [References](#). [Modifier &](#) and [Keyword this](#)

Function templates

[Overloaded functions](#) are commonly used to perform similar operations on various data types. [ArraySize\(\)](#) is a simple example of such function in MQL5. It returns size of any type of array. In fact, this system function is overloaded and the entire implementation of such an overload is hidden from MQL5 application developers:

```
int ArraySize(  
    void& array[]      // checked array  
);
```

It means that MQL5 language compiler inserts necessary implementation for each call of this function. For example, that is how it can be done for integer type arrays:

```
int ArraySize(  
    int& array[]       // array with int type elements  
);
```

[ArraySize\(\)](#) function can be displayed the following way for [MqlRates](#) type array for working with quotations in historical data format:

```
int ArraySize(  
    MqlRates& array[] // array filled with MqlRates type values  
);
```

Thus, it is very convenient to use the same function for working with different types. However, all preliminary work should be carried out - the necessary function should be [overloaded](#) for all data types it should correctly work with.

There is a convenient solution. If similar operations should be executed for each data type, it is possible to use function templates. In this case, a programmer needs to write only one function template description. When describing the template in such a way, we should specify only some formal parameter instead of some definite data type the function should work with. The compiler will automatically generate various functions for the appropriate handling of each type based on the types of the arguments used when calling the function.

Function template definition starts with the [template](#) keyword followed by the list of formal parameters in angle brackets. Each formal parameter is preceded by the [typename](#) keyword. Formal parameter types are built-in or user-defined types. They are used:

- to specify the types of function arguments,
- to specify the types of function's return value,
- to declare the variables inside the function definition

Number of template parameters cannot exceed eight. Each formal parameter in the template definition should appear in the list of function parameters at least once. Each name of the formal parameter should be unique.

Below is an example of a function template for searching the highest value in the array of any numeric type (integer and real numbers):


```

template<typename T>
T ArrayMax(T &arr[])
{
    uint size=ArraySize(arr);
    if(size==0) return(0);

    T max=arr[0];
    for(uint n=1;n<size;n++)
        if(max<arr[n]) max=arr[n];
    //--
    return(max);
}

```

This template defines the function that finds the highest value in the passed array and returns this value as a result. Keep in mind that the [ArrayMaximum\(\)](#) function built in MQL5 returns only the highest value index that can be used to find the value itself. For example:

```

/-- create an array
double array[];
int size=50;
ArrayResize(array,size);
/-- fill with random values
for(int i=0;i<size;i++)
{
    array[i]=MathRand();
}

/-- find position of the highest value in the array
int max_position=ArrayMaximum(array);
/-- now, get the highest value itself in the array
double max=array[max_position];
/-- display the found value
Print("Max value = ",max);

```

Thus, we have performed two steps to get the highest value in the array. With ArrayMax() function template, we can get the result of the necessary type just by passing the array of an appropriate type into this function. It means that instead of two last lines

```

/-- find position of the highest value in the array
int max_position=ArrayMaximum(array);
/-- now, get the highest value itself in the array
double max=array[max_position];

```

we now can use only one line, in which the returned result has the same type as the array passed into function:

```

/-- find the highest value
double max=ArrayMax(array);

```

In this case, the type of result returned by the ArrayMax() function will automatically match the type of array.

Use the [typename](#) keyword to get the argument type as a string in order to create general purpose methods of working with various data types. Let's consider a specific example of the function that returns data type as a string:

```
#include <Trade\Trade.mqh>
//+-----+
//|                                     |
//+-----+

void OnStart()
{
    //---
    CTrade trade;
    double d_value=M_PI;
    int i_value=INT_MAX;
    Print("d_value: type=",GetTypeName(d_value), ", value=", d_value);
    Print("i_value: type=",GetTypeName(i_value), ", value=", i_value);
    Print("trade: type=",GetTypeName(trade));
    //---
}

//+-----+
//| Type is returned as a line         |
//+-----+

template<typename T>
string GetTypeName(const T &t)
{
    //--- return the type as a line
    return(GetTypeName(T));
    //---
}
```

Function templates can also be used for class methods, for example:

```
class CFile
{
    ...
public:
    ...
    template<typename T>
    uint WriteStruct(T &data);
};

template<typename T>
uint CFile::WriteStruct(T &data)
{
    ...
    return(FileWriteStruct(m_handle,data));
}
```


Function templates should not be declared with [export](#), [virtual](#) and [#import](#) keywords.

Template function overload

A template function overload may be necessary sometimes. For example, we have a template function that writes the value of the second parameter to the first one using [typecasting](#). MQL5 does not allow typecasting [string](#) to [bool](#). We can do that ourselves - let's create an overload of a template function. For example:

```
//+-----+
//| Template function |
//+-----+
template<typename T1,typename T2>
string Assign(T1 &var1,T2 var2)
{
    var1=(T1)var2;
    return(__FUNCSIG__);
}
//+-----+
//| Special overload for bool+string |
//+-----+
string Assign(bool &var1,string var2)
{
    var1=(StringCompare(var2,"true",false) || StringToInteger(var2)!=0);
    return(__FUNCSIG__);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    int i;
    bool b;
    Print(Assign(i,"test"));
    Print(Assign(b,"test"));
}
```

As a result of the code execution, we can see that the Assign() template function has been used for the int+string pair, while the overloaded version has already been used for the bool+string pair during the second call.

```
string Assign<int,string>(int&,string)
string Assign(bool&,string)
```

See also

[Overload](#)

Template advantages

[Function templates](#) are used when you need to perform similar operations on various data types, for example, searching for a maximum element in the array. The main advantage of applying the templates is that you do not have to code a separate [overload](#) for each type. Instead of declaring multiple overloads of each type

```
double ArrayMax(double array[])
{
    ...
}
int ArrayMax(int array[])
{
    ...
}
uint ArrayMax(uint array[])
{
    ...
}
long ArrayMax(long array[])
{
    ...
}
datetime ArrayMax(datetime array[])
{
    ...
}
```

we need to write only one template function

```
template<typename T>
T ArrayMax(T array[])
{
    if(ArraySize()==0)
        return(0);
    uint max_index=ArrayMaximum(array);
    return(array[max_index]);
}
```

to use it in your code:

```
double high[];
datetime time[];
....
double max_high=ArrayMax(high);
datetime lasttime=ArrayMax(time);
```

Here, the *T* formal parameter specifying a type of used data is replaced with an actually applied type during compilation, i.e. the compiler automatically generates a separate function for each type - [double](#), [datetime](#), etc. MQL5 also allows you to develop class templates using all the advantages of the approach.

Class templates

A class template is declared using the **template** keyword followed by angle brackets<> enumerating the list of formal parameters with the **typename keyword**. This entry informs the compiler that it deals with a generic class with the *T* formal parameter defining a real variable type when implementing a class. For example, let's create a vector class for storing an array with *T* type elements:

```
#define TOSTR(x) #x+" " // macro for displaying an object name
//+-----+
//| Vector class for storing T-type elements |
//+-----+
template <typename T>
class TArray
{
protected:
    T          m_array[];
public:
    //--- constructor creates an array for 10 elements by default
    void TArray(void) {ArrayResize(m_array,10);}
    //--- constructor for creating a vector with a specified array size
    void TArray(int size) {ArrayResize(m_array,size);}
    //--- return a type and amount of data stored in the TArray type object
    string Type(void) {return(typename(m_array[0])+" "+(string)ArraySize(m_array));};
};
```

Next, let's apply different methods to create three *TArray* objects in the program for working with various types

```
void OnStart()
{
    TArray<double> double_array; // vector has a default size of 10
    TArray<int> int_array(15);   // vector has a size of 15
    TArray<string> *string_array; // pointer to TArray<string> vector
    //--- create a dynamic object
    string_array=new TArray<string>(20);
    //--- display an object name, data type and vector size in the Journal
    PrintFormat("%s (%s)",TOSTR(double_array),double_array.Type());
    PrintFormat("%s (%s)",TOSTR(int_array),int_array.Type());
    PrintFormat("%s (%s)",TOSTR(string_array),string_array.Type());
    //--- remove a dynamic object before completing the program
    delete(string_array);
}
```

Script execution results:

```
double_array (double:10)
int_array (int:15)
string_array (string:20)
```

Now, we have 3 vectors with different data types: double, int and string.

Class templates are well suited for developing containers - objects designed for encapsulating other objects of any type. Container objects are collections already containing objects of one certain type. Usually, working with stored data is instantly built into the container.

For example, you can create a class template that does not allow accessing an element outside the array, thus avoiding the "out of range" [critical error](#).

```
//+-----+
//| Class for a free access to an array element |
//+-----+
template<typename T>
class TSafeArray
{
protected:
    T          m_array[];
public:
    ///--- default constructor
    void        TSafeArray(void) {}
    ///--- constructor for creating the array of a specified size
    void        TSafeArray(int size) {ArrayResize(m_array,size);}
    ///--- array size
    int         Size(void) {return(ArraySize(m_array));}
    ///--- change the array size
    int         Resize(int size,int reserve) {return(ArrayResize(m_array,size,reserve));}
    ///--- release the array
    void        Erase(void) {ZeroMemory(m_array);}
    ///--- operator for accessing the array element by index
    T           operator[] (int index);
    ///--- assignment operator for receiving all elements from the array at once
    void        operator=(const T &array[]); // T type array
};
//+-----+
//| Receiving an element by index |
//+-----+
template<typename T>
T TSafeArray::operator[] (int index)
{
    static T invalid_value;
    ///---
    int max=ArraySize(m_array)-1;
    if(index<0 || index>=ArraySize(m_array))
    {
        PrintFormat("%s index %d is not in range (0-%d)!", __FUNCTION__, index, max);
        return(invalid_value);
    }
    ///---
    return(m_array[index]);
}
//+-----+
```



```

//| Assigning for the array |
//+-----+
template<typename T>
void TSafeArray::operator=(const T &array[])
{
    int size=ArraySize(array);
    ArrayResize(m_array,size);
//--- T type should support the copying operator
    for(int i=0;i<size;i++)
        m_array[i]=array[i];
//---
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    int copied,size=15;
    MqlRates rates[];
//--- copy the array of quotes
    if((copied=CopyRates(_Symbol,_Period,0,size,rates))!=size)
    {
        PrintFormat("CopyRates(%s,%s,0,%d) returned %d error code",
            _Symbol,EnumToString(_Period),size,GetLastError());
        return;
    }
//--- create a container and insert the MqlRates value array to it
    TSafeArray<MqlRates> safe_rates;
    safe_rates=rates;
//--- index within the array
    int index=3;
    PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
//--- index outside the array
    index=size;
    PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
}

```

Please note that template declaration should also be used when describing methods outside the class declaration:

```

template<typename T>
T TSafeArray::operator[](int index)
{
    ...
}
template<typename T>
void TSafeArray::operator=(const T &array[])
{
    ...
}

```



```
}
```

Class and function templates allow you to define multiple comma-separated formal parameters, for example, Map collection for storing "key - value" pairs:

```
template<typename Key, template Value>
class TMap
{
    ...
}
```

See also

[Function templates](#), [Overload](#)

Abstract Classes and Pure Virtual Functions

Abstract classes are used for creating generic entities, that you expect to use for creating more specific derived classes. An abstract class can only be used as the base class for some other class, that is why it is impossible to create an object of the abstract class type.

A class which contains at least one pure virtual function in it is abstract. Therefore, classes derived from the abstract class must implement all its pure virtual functions, otherwise they will also be abstract classes.

A virtual function is declared as "pure" by using the pure-specifier syntax. Consider the example of the CAnimal class, which is only created to provide common functions - the objects of the CAnimal type are too general for practical use. Thus, CAnimal is a good example for an abstract class:

```
class CAnimal
{
public:
    CAnimal();           // Constructor
    virtual void         Sound() = 0;    // A pure virtual function
private:
    double               m_legs_count;   // The number of the animal's legs
};
```

Here Sound() is a pure virtual function, because it is declared with the specifier of the pure virtual function PURE (=0).

Pure virtual functions are only the virtual functions for which the PURE specifier is set: (=NULL) or (=0). Example of abstract class declaration and use:

```
class CAnimal
{
public:
    virtual void         Sound()=NULL;    // PURE method, should be overridden in the derived class
};
//--- Derived from an abstract class
class CCat : public CAnimal
{
public:
    virtual void         Sound() { Print("Myau"); } // PURE is overridden, CCat is not abstract
};

//--- Examples of wrong use
new CAnimal;           // Error of 'CAnimal' - the compiler returns the "cannot instantiate abstract class"
CAnimal some_animal;    // Error of 'CAnimal' - the compiler returns the "cannot instantiate abstract class"

//--- Examples of proper use
new CCat;              // No error - the CCat class is not abstract
CCat cat;              // No error - the CCat class is not abstract
```


Restrictions on abstract classes

If the constructor for an abstract class calls a pure virtual function (either directly or indirectly), the result is undefined.

```
//+-----+
//| An abstract base class |
//+-----+
class CAnimal
{
public:
    ///--- A pure virtual function
    virtual void      Sound(void)=NULL;
    ///--- Function
    void              CallSound(void) { Sound(); }
    ///--- Constructor
    CAnimal()
    {
        ///--- An explicit call of the virtual method
        Sound();
        ///--- An implicit call (using a third function)
        CallSound();
        ///--- A constructor and/or destructor always calls its own functions,
        ///--- even if they are virtual and overridden by a called function in a derived class
        ///--- If the called function is pure virtual,
        ///--- its call will cause a critical runtime error: "pure virtual function call"
    }
};
```

However, constructors and destructors for abstract classes can call other member functions.

Standard Constants, Enumerations and Structures

To simplify the program writing and to make program texts more convenient for perception, the MQL5 language provides predefined standard constants and enumerations. Besides that, service [structures](#) are used for storing information.

Standard constants are similar to macros and are of [int](#) type.

The constants are grouped by their purposes:

- [Chart constants](#) are used when working with price charts: opening, navigation, setting parameters;
- [Objects constants](#) are intended for processing graphical objects that can be created and displayed in charts;
- [Indicators constants](#) are used for working with standard and custom indicators;
- [Environment state](#) constants describe properties of a MQL5-program, show information about a client terminal, financial instrument and current account;
- [Trade constants](#) allow to specify a variety of information in the course of trading;
- [Named constants](#) are constants of the MQL5 language;
- [Data structures](#) describe data storage formats used;
- [Codes of errors and warnings](#) describe compiler messages and trading server answers to trade requests;
- [In/out constants](#) are designed for working with [file functions](#) and displaying messages on the screen by the [MessageBox\(\)](#) function.

Chart Constants

Constants describing various properties of charts are divided into the following groups:

- [Types of events](#) - events that occur when working with charts;
- [Chart timeframes](#) - standard built-in periods;
- [Properties of chart](#) - identifiers that are used as parameters of [chart functions](#);
- [Positioning constants](#) - value of a parameter of the [ChartNavigate\(\)](#) function;
- [Displaying charts](#) - setting the chart appearance.

Types of Chart Events

There are 9 types of events that can be processed using the predefined function [OnChartEvent\(\)](#). For custom events 65535 identifiers are provided in the range of CHARTEVENT_CUSTOM to CHARTEVENT_CUSTOM_LAST inclusive. To generate a custom event, the [EventChartCustom\(\)](#) function should be used.

ENUM_CHART_EVENT

ID	Description
CHARTEVENT_KEYDOWN	Keystrokes
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE =true is set for the chart)
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT_CREATE =true is set for the chart)
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE =true is set for the chart)
CHARTEVENT_CLICK	Clicking on a chart
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit
CHARTEVENT_CHART_CHANGE	Change of the chart size or modification of chart properties through the Properties dialog
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events

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

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of)	The string value of a bit mask

			times the keystroke is repeated as a result of the user holding down the key)	describing the status of keyboard buttons
Mouse events (if CHART_EVENT_MOUSE_MOVE =true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
event of graphical object creation (if CHART_EVENT_OBJECT_CREATE =true is set for the chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE =true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text editing in the entry box of the	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit graphical object, in which text

LabelEdit graphical object				editing has completed
Event of change of the chart size or modification of chart properties through the Properties dialog	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

Example:

```

#define KEY_NUMPAD_5      12
#define KEY_LEFT         37
#define KEY_UP           38
#define KEY_RIGHT        39
#define KEY_DOWN         40
#define KEY_NUMLOCK_DOWN 98
#define KEY_NUMLOCK_LEFT 100
#define KEY_NUMLOCK_5    101
#define KEY_NUMLOCK_RIGHT 102
#define KEY_NUMLOCK_UP   104

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //---
    Print("The expert with name ",MQL5InfoString(MQL5_PROGRAM_NAME)," is running");
    //--- enable object create events
    ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
    //--- enable object delete events
    ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| ChartEvent function |
//+-----+
void OnChartEvent(const int id,          // Event identifier
                  const long& lparam,    // Event parameter of long type
                  const double& dparam,  // Event parameter of double type
                  const string& sparam   // Event parameter of string type
                  )
{

```



```

//--- the left mouse button has been pressed on the chart
if(id==CHARTEVENT_CLICK)
{
    Print("The coordinates of the mouse click on the chart are: x = ",lparam," y =
}
//--- the mouse has been clicked on the graphic object
if(id==CHARTEVENT_OBJECT_CLICK)
{
    Print("The mouse has been clicked on the object with name '"+sparam+"'");
}
//--- the key has been pressed
if(id==CHARTEVENT_KEYDOWN)
{
    switch(lparam)
    {
        case KEY_NUMLOCK_LEFT: Print("The KEY_NUMLOCK_LEFT has been pressed"); bre
        case KEY_LEFT:         Print("The KEY_LEFT has been pressed");         bre
        case KEY_NUMLOCK_UP:   Print("The KEY_NUMLOCK_UP has been pressed");   bre
        case KEY_UP:           Print("The KEY_UP has been pressed");             bre
        case KEY_NUMLOCK_RIGHT: Print("The KEY_NUMLOCK_RIGHT has been pressed"); bre
        case KEY_RIGHT:        Print("The KEY_RIGHT has been pressed");          bre
        case KEY_NUMLOCK_DOWN: Print("The KEY_NUMLOCK_DOWN has been pressed");   bre
        case KEY_DOWN:         Print("The KEY_DOWN has been pressed");           bre
        case KEY_NUMPAD_5:     Print("The KEY_NUMPAD_5 has been pressed");       bre
        case KEY_NUMLOCK_5:    Print("The KEY_NUMLOCK_5 has been pressed");      bre
        default:               Print("Some not listed key has been pressed");
    }
    ChartRedraw();
}
//--- the object has been deleted
if(id==CHARTEVENT_OBJECT_DELETE)
{
    Print("The object with name ",sparam," has been deleted");
}
//--- the object has been created
if(id==CHARTEVENT_OBJECT_CREATE)
{
    Print("The object with name ",sparam," has been created");
}
//--- the object has been moved or its anchor point coordinates has been changed
if(id==CHARTEVENT_OBJECT_DRAG)
{
    Print("The anchor point coordinates of the object with name ",sparam," has been
}
//--- the text in the Edit of object has been changed
if(id==CHARTEVENT_OBJECT_ENDEDIT)
{
    Print("The text in the Edit field of the object with name ",sparam," has been ch
}

```



```
}
```

For CHARTEVENT_MOUSE_MOVE event the **sparam** string parameter contains information about state of the keyboard and mouse buttons:

Bit	Description
1	State of the left mouse button
2	State of the right mouse button
3	State of the SHIFT button
4	State of the CTRL button
5	State of the middle mouse button
6	State of the first extra mouse button
7	State of the second extra mouse button

Example:

```
//+-----+
//| Expert initialization function |
//+-----+
void OnInit()
{
    //--- enable CHART_EVENT_MOUSE_MOVE messages
    ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
}
//+-----+
//| MouseState |
//+-----+
string MouseState(uint state)
{
    string res;
    res+="\nML: " +(((state& 1)== 1)?"DN":"UP"); // mouse left
    res+="\nMR: " +(((state& 2)== 2)?"DN":"UP"); // mouse right
    res+="\nMM: " +(((state&16)==16)?"DN":"UP"); // mouse middle
    res+="\nMX: " +(((state&32)==32)?"DN":"UP"); // mouse first X key
    res+="\nMY: " +(((state&64)==64)?"DN":"UP"); // mouse second X key
    res+="\nSHIFT: " +(((state& 4)== 4)?"DN":"UP"); // shift key
    res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP"); // control key
    return(res);
}
//+-----+
//| ChartEvent function |
//+-----+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &sparam)
{
    if(id==CHARTEVENT_MOUSE_MOVE)
```



```
Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));  
}
```

See also

[Event Handling Functions](#), [Working with events](#)

Chart Timeframes

All predefined timeframes of charts have unique identifiers. The `PERIOD_CURRENT` identifier means the current period of a chart, at which a mql5-program is running.

ENUM_TIMEFRAMES

ID	Description
PERIOD_CURRENT	Current timeframe
PERIOD_M1	1 minute
PERIOD_M2	2 minutes
PERIOD_M3	3 minutes
PERIOD_M4	4 minutes
PERIOD_M5	5 minutes
PERIOD_M6	6 minutes
PERIOD_M10	10 minutes
PERIOD_M12	12 minutes
PERIOD_M15	15 minutes
PERIOD_M20	20 minutes
PERIOD_M30	30 minutes
PERIOD_H1	1 hour
PERIOD_H2	2 hours
PERIOD_H3	3 hours
PERIOD_H4	4 hours
PERIOD_H6	6 hours
PERIOD_H8	8 hours
PERIOD_H12	12 hours
PERIOD_D1	1 day
PERIOD_W1	1 week
PERIOD_MN1	1 month

Example:

```
string chart_name="test_Object_Chart";
Print("Let's try to create a Chart object with the name ",chart_name);
//--- If such an object does not exist - create it
if(ObjectFind(0,chart_name)<0)ObjectCreate(0,chart_name,OBJ_CHART,0,0,0,0,0);
```



```
//--- Define symbol
ObjectSetString(0,chart_name,OBJPROP_SYMBOL,"EURUSD");
//--- Set X coordinate of the anchor point
ObjectSetInteger(0,chart_name,OBJPROP_XDISTANCE,100);
//--- Set Y coordinate of the anchor point
ObjectSetInteger(0,chart_name,OBJPROP_YDISTANCE,100);
//--- Set the width of chart
ObjectSetInteger(0,chart_name,OBJPROP_XSIZE,400);
//--- Set the height
ObjectSetInteger(0,chart_name,OBJPROP_YSIZE,300);
//--- Set the timeframe
ObjectSetInteger(0,chart_name,OBJPROP_PERIOD,PERIOD_D1);
//--- Set scale (from 0 to 5)
ObjectSetDouble(0,chart_name,OBJPROP_SCALE,4);
//--- Disable selection by a mouse
ObjectSetInteger(0,chart_name,OBJPROP_SELECTABLE,false);
```

See also

[PeriodSeconds](#), [Period](#), [Date and Time](#), [Visibility of objects](#)

Chart Properties

Identifiers of ENUM_CHART_PROPERTY enumerations are used as parameters of [functions for working with charts](#). The abbreviation of r/o in the "Property Type" column means that this property is read-only and cannot be changed. The w/o abbreviation in the "Property Type" column means that this property is write-only and it cannot be received. When accessing certain properties, it's necessary to specify an additional parameter-modifier (modifier), which serves to indicate the number of chart subwindows. 0 means the main window.

The functions defining the chart properties are actually used for sending change commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. The changes are implemented to the chart when handling the queue of the chart events.

Thus, do not expect an immediate visual update of the chart after calling these functions. Generally, the chart is updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use [ChartRedraw\(\)](#) function to forcefully update the chart.

For functions [ChartSetInteger\(\)](#) and [ChartGetInteger\(\)](#)

ENUM_CHART_PROPERTY_INTEGER

ID	Description	Property Type
CHART_SHOW	Price chart drawing. If false, drawing any price chart attributes is disabled and all chart border indents are eliminated, including time and price scales, quick navigation bar, Calendar event labels, trade labels, indicator and bar tooltips, indicator subwindows, volume histograms, etc. Disabling the drawing is a perfect solution for creating a custom program interface using the graphical resources . The graphical objects are always drawn regardless of the CHART_SHOW property value.	bool
CHART_IS_OBJECT	Identifying "Chart" (OBJ_CHART) object - returns true for a graphical object. Returns false for a real chart	bool r/o
CHART_BRING_TO_TOP	Show chart on top of other charts	bool w/o

<u>CHART_MOUSE_SCROLL</u>	Scrolling the chart horizontally using the left mouse button. Vertical scrolling is also available if the value of any following properties is set to true: <u>CHART_SCALEFIX</u> , <u>CHART_SCALEFIX_11</u> or <u>CHART_SCALE_PT_PER_BAR</u>	bool
<u>CHART_EVENT_MOUSE_MOVE</u>	Send notifications of mouse move and mouse click events (<u>CHARTEVENT_MOUSE_MOVE</u>) to all mql5 programs on a chart	bool
<u>CHART_EVENT_OBJECT_CREATE</u>	Send a notification of an event of new object creation (<u>CHARTEVENT_OBJECT_CREATE</u>) to all mql5-programs on a chart	bool
<u>CHART_EVENT_OBJECT_DELETE</u>	Send a notification of an event of object deletion (<u>CHARTEVENT_OBJECT_DELETE</u>) to all mql5-programs on a chart	bool
<u>CHART_MODE</u>	Chart type (candlesticks, bars or line)	enum <u>ENUM_CHART_MODE</u>
<u>CHART_FOREGROUND</u>	Price chart in the foreground	bool
<u>CHART_SHIFT</u>	Mode of price chart indent from the right border	bool
<u>CHART_AUTOSCROLL</u>	Mode of automatic moving to the right border of the chart	bool
<u>CHART_KEYBOARD_CONTROL</u>	Allow managing the chart using a keyboard ("Home", "End", "PageUp", "+", "-", "Up arrow", etc.). Setting <u>CHART_KEYBOARD_CONTROL</u> to false disables chart scrolling and scaling while leaving intact the ability to receive the keys pressing events in <u>OnChartEvent()</u> .	bool
<u>CHART_QUICK_NAVIGATION</u>	Allow the chart to intercept Space and Enter key strokes to activate the quick navigation bar. The quick navigation bar automatically appears at the bottom of the chart after	bool

	double-clicking the mouse or pressing Space/Enter. It allows you to quickly change a symbol, timeframe and first visible bar date.	
<u>CHART_SCALE</u>	Scale	int from 0 to 5
<u>CHART_SCALEFIX</u>	Fixed scale mode	bool
<u>CHART_SCALEFIX_11</u>	Scale 1:1 mode	bool
<u>CHART_SCALE_PT_PER_BAR</u>	Scale to be specified in points per bar	bool
<u>CHART_SHOW_OHLC</u>	Show OHLC values in the upper left corner	bool
<u>CHART_SHOW_BID_LINE</u>	Display Bid values as a horizontal line in a chart	bool
<u>CHART_SHOW_ASK_LINE</u>	Display Ask values as a horizontal line in a chart	bool
<u>CHART_SHOW_LAST_LINE</u>	Display Last values as a horizontal line in a chart	bool
<u>CHART_SHOW_PERIOD_SEP</u>	Display vertical separators between adjacent periods	bool
<u>CHART_SHOW_GRID</u>	Display grid in the chart	bool
<u>CHART_SHOW_VOLUMES</u>	Display volume in the chart	enum <u>ENUM_CHART_VOLUME_MODE</u>
<u>CHART_SHOW_OBJECT_DESCR</u>	Display textual descriptions of objects (not available for all objects)	bool
<u>CHART_VISIBLE_BARS</u>	The number of bars on the chart that can be displayed	int r/o
<u>CHART_WINDOWS_TOTAL</u>	The total number of chart windows, including indicator subwindows	int r/o
<u>CHART_WINDOW_IS_VISIBLE</u>	Visibility of subwindows	bool r/o modifier - subwindow number
<u>CHART_WINDOW_HANDLE</u>	Chart window handle (HWND)	int r/o
<u>CHART_WINDOW_YDISTANCE</u>	The distance between the upper frame of the indicator subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a	int r/o modifier - subwindow number

	<p>mouse event, the cursor coordinates are passed in terms of the coordinates of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow.</p> <p>The value is required for converting the absolute coordinates of the main chart to the local coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.</p>	
<u>CHART_FIRST_VISIBLE_BAR</u>	Number of the first visible bar in the chart. Indexing of bars is the same as for <u>timeseries</u> .	int r/o
<u>CHART_WIDTH_IN_BARS</u>	Chart width in bars	int r/o
<u>CHART_WIDTH_IN_PIXELS</u>	Chart width in pixels	int r/o
<u>CHART_HEIGHT_IN_PIXELS</u>	Chart height in pixels	int modifier - subwindow number
<u>CHART_COLOR_BACKGROUND</u>	Chart background color	color
<u>CHART_COLOR_FOREGROUND</u>	Color of axes, scales and OHLC line	color
<u>CHART_COLOR_GRID</u>	Grid color	color
<u>CHART_COLOR_VOLUME</u>	Color of volumes and position opening levels	color
<u>CHART_COLOR_CHART_UP</u>	Color for the up bar, shadows and body borders of bull candlesticks	color
<u>CHART_COLOR_CHART_DOWN</u>	Color for the down bar, shadows and body borders of bear candlesticks	color
<u>CHART_COLOR_CHART_LINE</u>	Line chart color and color of "Doji" Japanese candlesticks	color
<u>CHART_COLOR_CANDLE_BULL</u>	Body color of a bull candlestick	color
<u>CHART_COLOR_CANDLE_BEAR</u>	Body color of a bear candlestick	color

<u>CHART_COLOR_BID</u>	Bid price level color	color
<u>CHART_COLOR_ASK</u>	Ask price level color	color
<u>CHART_COLOR_LAST</u>	Line color of the last executed deal price (Last)	color
<u>CHART_COLOR_STOP_LEVEL</u>	Color of stop order levels (Stop Loss and Take Profit)	color
<u>CHART_SHOW_TRADE_LEVELS</u>	Displaying trade levels in the chart (levels of open positions, Stop Loss, Take Profit and pending orders)	bool
<u>CHART_DRAG_TRADE_LEVELS</u>	Permission to drag trading levels on a chart with a mouse. The drag mode is enabled by default (true value)	bool
<u>CHART_SHOW_DATE_SCALE</u>	Showing the time scale on a chart	bool
<u>CHART_SHOW_PRICE_SCALE</u>	Showing the price scale on a chart	bool
<u>CHART_SHOW_ONE_CLICK</u>	Showing the " <u>One click trading</u> " panel on a chart	bool
<u>CHART_IS_MAXIMIZED</u>	Chart window is maximized	bool
<u>CHART_IS_MINIMIZED</u>	Chart window is minimized	bool

For functions [ChartSetDouble\(\)](#) and [ChartGetDouble\(\)](#)

ENUM_CHART_PROPERTY_DOUBLE

ID	Description	Property Type
<u>CHART_SHIFT_SIZE</u>	The size of the zero bar indent from the right border in percents	double (from 10 to 50 percents)
<u>CHART_FIXED_POSITION</u>	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the horizontal time axis. It is displayed only if the automatic chart scrolling to the right on tick incoming is disabled (see <u>CHART_AUTOSCROLL</u> property). The bar on a fixed position remains in the same place when zooming in and out.	double

<u>CHART_FIXED_MAX</u>	Fixed chart maximum	double
<u>CHART_FIXED_MIN</u>	Fixed chart minimum	double
<u>CHART_POINTS_PER_BAR</u>	Scale in points per bar	double
<u>CHART_PRICE_MIN</u>	Chart minimum	double r/o modifier - subwindow number
<u>CHART_PRICE_MAX</u>	Chart maximum	double r/o modifier - subwindow number

For functions [ChartSetString\(\)](#) and [ChartGetString\(\)](#)

ENUM_CHART_PROPERTY_STRING

ID	Description	Property Type
<u>CHART_COMMENT</u>	Text of a comment in a chart	string
CHART_EXPERT_NAME	The name of the Expert Advisor running on the chart with the specified chart_id	string
CHART_SCRIPT_NAME	The name of the script running on the chart with the specified chart_id	string

Example:

```
int chartMode=ChartGetInteger(0,CHART_MODE);
switch(chartMode)
{
    case(CHART_BARS):    Print("CHART_BARS");    break;
    case(CHART_CANDLES): Print("CHART_CANDLES");break;
    default:Print("CHART_LINE");
}
bool shifted=ChartGetInteger(0,CHART_SHIFT);
if(shifted) Print("CHART_SHIFT = true");
else Print("CHART_SHIFT = false");
bool autoscroll=ChartGetInteger(0,CHART_AUTOSCROLL);
if(autoscroll) Print("CHART_AUTOSCROLL = true");
else Print("CHART_AUTOSCROLL = false");
int chartHandle=ChartGetInteger(0,CHART_WINDOW_HANDLE);
Print("CHART_WINDOW_HANDLE = ",chartHandle);
int windows=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
Print("CHART_WINDOWS_TOTAL = ",windows);
if(windows>1)
{
    for(int i=0;i<windows;i++)
    {
        int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,i);
        double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,i);
```



```
double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,i);  
Print(i+": CHART_HEIGHT_IN_PIXELS = ",height," pixels");  
Print(i+": CHART_PRICE_MIN = ",priceMin);  
Print(i+": CHART_PRICE_MAX = ",priceMax);  
}  
}
```

See also

[Examples of Working with the Chart](#)

Positioning Constants

Three identifiers from the ENUM_CHART_POSITION list are the possible values of the *position* parameter for the [ChartNavigate\(\)](#) function.

ENUM_CHART_POSITION

ID	Description
CHART_BEGIN	Chart beginning (the oldest prices)
CHART_CURRENT_POS	Current position
CHART_END	Chart end (the latest prices)

Example:

```
long handle=ChartOpen("EURUSD",PERIOD_H12);
if(handle!=0)
{
    ChartSetInteger(handle,CHART_AUTOSCROLL,false);
    ChartSetInteger(handle,CHART_SHIFT,true);
    ChartSetInteger(handle,CHART_MODE,CHART_LINE);
    ResetLastError();
    bool res=ChartNavigate(handle,CHART_END,150);
    if(!res) Print("Navigate failed. Error = ",GetLastError());
    ChartRedraw();
}
```


Chart Representation

Price charts can be displayed in three ways:

- as bars;
- as candlesticks;
- as a line.

The specific way of displaying the price chart is set by the function [ChartSetInteger](#)(chart_handle, [CHART_MODE](#), chart_mode), where chart_mode is one of the values of the ENUM_CHART_MODE enumeration.

ENUM_CHART_MODE

ID	Description
CHART_BARS	Display as a sequence of bars
CHART_CANDLES	Display as Japanese candlesticks
CHART_LINE	Display as a line drawn by Close prices

To specify the mode of displaying volumes in the price chart the function [ChartSetInteger](#)(chart_handle, [CHART_SHOW_VOLUMES](#), volume_mode) is used, where volume_mode is one of values of the ENUM_CHART_VOLUME_MODE enumeration.

ENUM_CHART_VOLUME_MODE

ID	Description
CHART_VOLUME_HIDE	Volumes are not shown
CHART_VOLUME_TICK	Tick volumes
CHART_VOLUME_REAL	Trade volumes

Example:

```
//--- Get the handle of the current chart
long handle=ChartID();
if(handle>0) // If it succeeded, additionally customize
{
    //--- Disable autoscroll
    ChartSetInteger(handle,CHART_AUTOSCROLL,false);
    //--- Set the indent of the right border of the chart
    ChartSetInteger(handle,CHART_SHIFT,true);
    //--- Display as candlesticks
    ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
    //--- Scroll by 100 bars from the beginning of history
    ChartNavigate(handle,CHART_CURRENT_POS,100);
    //--- Set the tick volume display mode
    ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
}
```



```
}
```

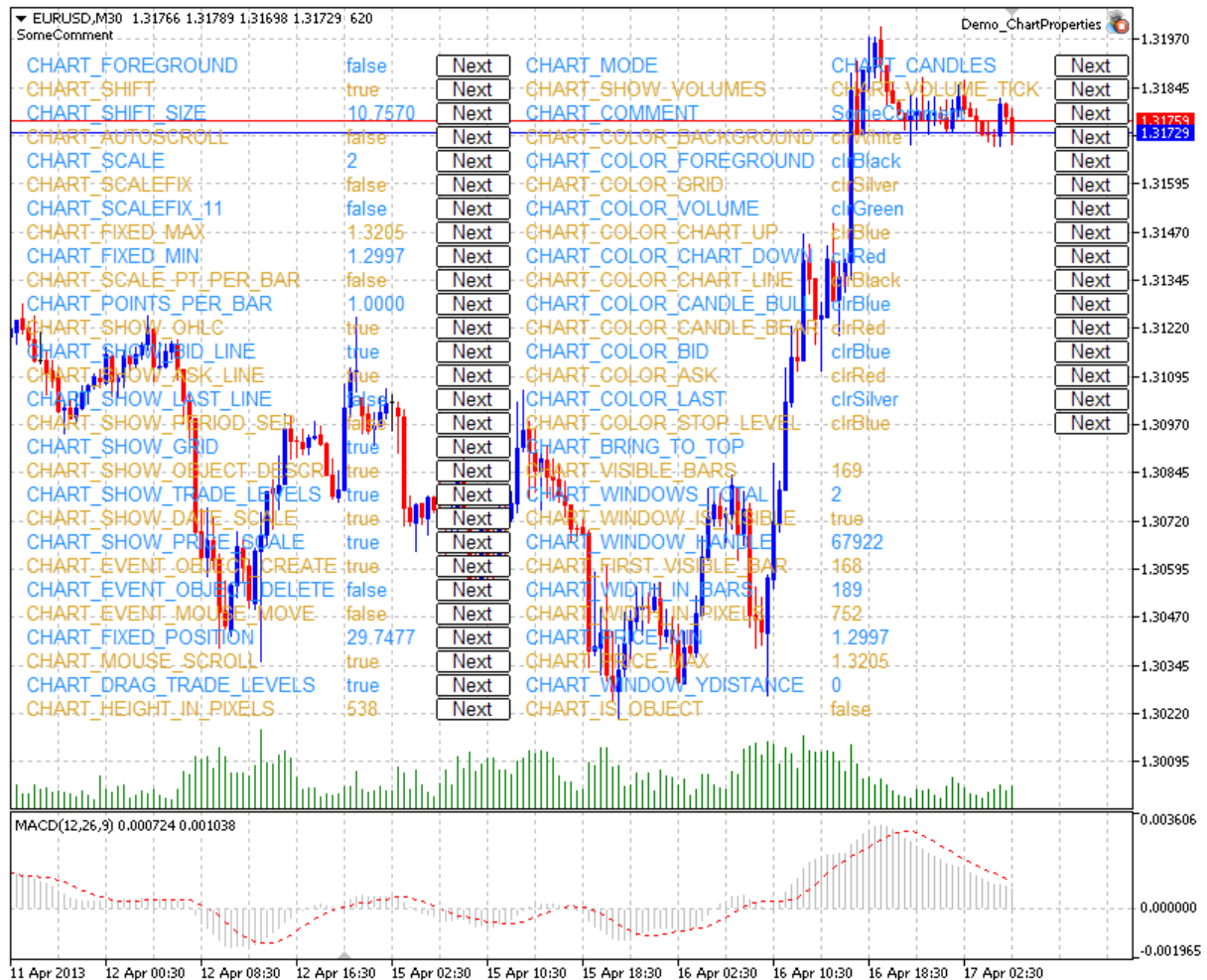
See also

[ChartOpen](#), [ChartID](#)

Examples of Working with the Chart

This section contains examples of working with chart properties. One or two complete functions are displayed for each property. These functions allow setting/receiving the value of the property. These functions can be used "as is" in custom mql5 applications.

The screenshot below demonstrates the graphic panel illustrating how changing of [the chart property](#) changes its appearance. Clicking Next button allows setting the new value of the appropriate property and view the changes in the chart window.



The panel's source code is located [below](#).

Chart Properties and Sample Functions for Working with Them

- **CHART_IS_OBJECT** defines if an object is a real chart or a [graphic object](#).

```
//+-----+
//| Checks if an object is a chart. If it is a graphic object, |
//| the result is true. If it is a real chart, the result variable |
//| has the value of false. |
//+-----+

bool ChartIsObject(bool &result,const long chart_ID=0)
{
```



```

//--- prepare the variable to get the property value
long value;
//--- reset the error value
ResetLastError();
//--- get the chart property
if(!ChartGetInteger(chart_ID, CHART_IS_OBJECT, 0, value))
{
    //--- display the error message in Experts journal
    Print(__FUNCTION__ + ", Error Code = ", GetLastError());
    //--- return false
    return(false);
}
//--- store the value of the chart property in memory
result=value;
//--- successful execution
return(true);
}

```

- **CHART_BRING_TO_TOP** shows the chart on top of all others.

```

//+-----+
//| Sends command to the terminal to display the chart above all others |
//+-----+
bool ChartBringToTop(const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- show the chart on top of all others
    if(!ChartSetInteger(chart_ID, CHART_BRING_TO_TOP, 0, true))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__ + ", Error Code = ", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_MOUSE_SCROLL** is a property for scrolling the chart using left mouse button.

```

//+-----+
//| Checks if scrolling of chart using left mouse button is enabled |
//+-----+
bool ChartMouseScrollGet(bool &result, const long chart_ID=0)
{

```



```

//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}

//+-----+
//| Enables/disables scrolling of chart using left mouse button |
//+-----+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_EVENT_MOUSE_MOVE** is a property of sending messages concerning move events and mouse clicks to mql5 applications ([CHARTEVENT_MOUSE_MOVE](#)).

```

//+-----+
//| Checks if messages concerning move events and mouse clicks |
//| are sent to all MQL5 applications on the chart |
//+-----+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value

```



```

    if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

//+-----+
//| Enables/disables the mode of sending messages concerning move events and |
//| mouse clicks to MQL5 applications on the chart                          |
//+-----+
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_EVENT_OBJECT_CREATE** is a property of sending messages concerning the event of a graphic object creation to mql5 applications ([CHARTEVENT_OBJECT_CREATE](#)).

```

//+-----+
//| Checks if messages concerning the event of a graphic                    |
//| object creation are sent to all MQL5 applications on the chart          |
//+-----+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
    }
}

```



```

        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}
//+-----+
//| Enables/disables the mode of sending messages concerning the event of a |
//| graphic object creation to all mql5 applications on the chart           |
//+-----+
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_EVENT_OBJECT_DELETE** is a property of sending messages concerning the event of a graphic object deletion to mql5 applications ([CHARTEVENT_OBJECT_DELETE](#)).

```

//+-----+
//| Checks if messages concerning the event of a graphic object           |
//| deletion are sent to all mql5 applications on the chart               |
//+-----+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
}

```



```
//--- successful execution
    return(true);
}

//+-----+
//| Enables/disables the mode of sending messages concerning the event of a |
//| graphic object deletion to all mql5 applications on the chart          |
//+-----+
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
```

- **CHART_MODE** - type of the chart (candlesticks, bars or line).

```
//+-----+
//| Gets chart display type (candlesticks, bars or line)                  |
//+-----+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long result=WRONG_VALUE;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((ENUM_CHART_MODE)result);
}

//+-----+
//| Sets chart display type (candlesticks, bars or line)                  |
//+-----+
bool ChartModeSet(const long value,const long chart_ID=0)
{
//--- reset the error value
```



```

    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_MODE,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_FOREGROUND** is a property of displaying a price chart in the foreground.

```

//+-----+
//| Checks if a price chart is displayed in the foreground |
//+-----+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of a price chart on the foreground |
//+-----+
bool ChartForegroundSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
}

```



```

    }
    //--- successful execution
    return(true);
}

```

- **CHART_SHIFT** - mode of shift of the price chart from the right border.

```

//+-----+
//| Checks if shifting a price chart from the right border is enabled |
//+-----+
bool ChartShiftGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

//+-----+
//| Enables/disables displaying of a price chart with a shift from the right border |
//+-----+
bool ChartShiftSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```


- **CHART_AUTOSCROLL** - the mode of automatic shift to the right border of the chart.

```
//+-----+
//| Checks if automatic scrolling of a chart to the right |
//| on new ticks arrival is enabled |
//+-----+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables automatic scrolling of a chart to the right |
//| on new ticks arrival |
//+-----+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
```

- **CHART_SCALE** - chart scale property.


```

//+-----+
//| Gets chart scale (from 0 to 5) |
//+-----+
int ChartScaleGet(const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long result=-1;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((int)result);
}
//+-----+
//| Sets chart scale (from 0 to 5) |
//+-----+
bool ChartScaleSet(const long value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SCALEFIX** - the mode of fixed chart scale.

```

//+-----+
//| Checks if the fixed scale mode is enabled |
//+-----+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value

```



```

    if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}
//+-----+
//| Enables/disables the fixed scale mode |
//+-----+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- CHART_SCALEFIX_11 - 1:1 chart scale mode.

```

//+-----+
//| Checks if the "1:1" scale is enabled |
//+-----+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory

```



```

    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables the "1:1" scale mode |
//+-----+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SCALE_PT_PER_BAR** - the mode of specifying the chart scale in points per bar.

```

//+-----+
//| Checks if the "points per bar" chart scaling mode is enabled |
//+-----+
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables the "points per bar" chart scaling mode |
//+-----+

```



```

bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_SHOW_OHLC** - the property of displaying OHLC values in the upper left corner.

```

//+-----+
//| Checks if displaying of OHLC values in the upper left corner of chart is enabled
//+-----+
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

//+-----+
//| Enables/disables displaying of OHLC values in the upper left corner of chart |
//+-----+
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
    {

```



```

    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = ", GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

```

- **CHART_SHOW_BID_LINE** - the property of displaying Bid value as a horizontal line on the chart.

```

//+-----+
//| Checks if displaying of Bid line on chart is enabled |
//+-----+
bool ChartShowBidLineGet(bool &result, const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID, CHART_SHOW_BID_LINE, 0, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ", GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of Bid line on chart |
//+-----+
bool ChartShowBidLineSet(const bool value, const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID, CHART_SHOW_BID_LINE, 0, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```


- **CHART_SHOW_ASK_LINE** - the property of displaying Ask value as a horizontal line on a chart.

```
//+-----+
//| Checks if displaying of Ask line on chart is enabled |
//+-----+
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}

//+-----+
//| Enables/disables displaying of Ask line on chart |
//+-----+
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
```

- **CHART_SHOW_LAST_LINE** - the property of displaying Last value as a horizontal line on a chart.

```
//+-----+
//| Checks if displaying of line for the last performed deal's price is enabled |
//+-----+
```



```

bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

//+-----+
//| Enables/disables displaying of line for the last performed deal's price |
//+-----+
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_SHOW_PERIOD_SEP** - the property of displaying vertical separators between adjacent periods.

```

//+-----+
//| Checks if displaying of vertical separators between adjacent periods is enabled |
//+-----+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();

```



```

//--- receive the property value
if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
{
    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = ",GetLastError());
    return(false);
}

//--- store the value of the chart property in memory
result=value;
//--- successful execution
return(true);
}

//+-----+
//| Enables/disables displaying of vertical separators between adjacent periods |
//+-----+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_SHOW_GRID** - the property of displaying the chart grid.

```

//+-----+
//| Checks if the chart grid is displayed |
//+-----+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
}

```



```

//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of grid on chart |
//+-----+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set the property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SHOW_VOLUMES** - the property of displaying the volumes on a chart.

```

//+-----+
//| Checks if volumes are displayed on a chart |
//| The flag indicates the volumes showing mode |
//+-----+
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long result=WRONG_VALUE;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((ENUM_CHART_VOLUME_MODE)result);
}
//+-----+
//| Sets mode of displaying volumes on chart |
//+-----+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)

```



```

{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID, CHART_SHOW_VOLUMES, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__ + ", Error Code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SHOW_OBJECT_DESCR** - the property of graphical object pop-up descriptions.

```

//+-----+
//| Checks if pop-up descriptions of graphical objects are displayed |
//| when hovering mouse over them                                     |
//+-----+
bool ChartShowObjectDescriptionGet(bool &result, const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID, CHART_SHOW_OBJECT_DESCR, 0, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__ + ", Error Code = ", GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of pop-up descriptions of graphical objects |
//| when hovering mouse over them                                     |
//+-----+
bool ChartShowObjectDescriptionSet(const bool value, const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();

```



```

//--- set property value
if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
{
    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

```

- **CHART_VISIBLE_BARS** defines the number of bars on a chart that are available for display.

```

//+-----+
//| Gets the number of bars that are displayed (visible) in chart window |
//+-----+
int ChartVisibleBars(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((int)result);
}

```

- **CHART_WINDOWS_TOTAL** defines the total number of chart windows including indicator subwindows.

```

//+-----+
//| Gets the total number of chart windows including indicator subwindows |
//+-----+
int ChartWindowsTotal(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
    {

```



```

    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = "", GetLastError());
}
//--- return the value of the chart property
return((int)result);
}

```

- **CHART_WINDOW_IS_VISIBLE** defines the subwindow's visibility.

```

//+-----+
//| Checks if the current chart window or subwindow is visible |
//+-----+
bool ChartWindowsIsVisible(bool &result, const long chart_ID=0, const int sub_window=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID, CHART_WINDOW_IS_VISIBLE, sub_window, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

```

- **CHART_WINDOW_HANDLE** returns the chart handle.

```

//+-----+
//| Gets the chart handle |
//+-----+
int ChartWindowsHandle(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID, CHART_WINDOW_HANDLE, 0, result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
    }
}

```



```

    }
    //--- return the value of the chart property
    return((int)result);
}

```

- **CHART_WINDOW_YDISTANCE** defines the distance in pixels between the upper frame of the indicator subwindow and the upper frame of the chart's main window.

```

//+-----+
//| Gets the distance in pixels between the upper border of          |
//| subwindow and the upper border of chart's main window          |
//+-----+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((int)result);
}

```

- **CHART_FIRST_VISIBLE_BAR** returns the number of the first visible bar on the chart (bar indexing corresponds to the [time series](#)).

```

//+-----+
//| Gets the index of the first visible bar on chart.              |
//| Indexing is performed like in timeseries: latest bars have smallest indices. |
//+-----+
int ChartFirstVisibleBar(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_FIRST_VISIBLE_BAR,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
}

```



```
//--- return the value of the chart property
    return((int)result);
}
```

- **CHART_WIDTH_IN_BARS** returns the chart width in bars.

```
//+-----+
//| Gets the width of chart (in bars) |
//+-----+
int ChartWidthInBars(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((int)result);
}
```

- **CHART_WIDTH_IN_PIXELS** returns the chart width in pixels.

```
//+-----+
//| Gets the width of chart (in pixels) |
//+-----+
int ChartWidthInPixels(const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((int)result);
}
```


- **CHART_HEIGHT_IN_PIXELS** - chart height property in pixels.

```
//+-----+
//| Gets the height of chart (in pixels) |
//+-----+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
{
    //--- prepare the variable to get the property value
    long result=-1;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((int)result);
}

//+-----+
//| Sets the height of chart (in pixels) |
//+-----+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const int sub_window=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_COLOR_BACKGROUND** - chart background color.

```
//+-----+
//| Gets the background color of chart |
//+-----+
color ChartBackColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
```



```

    ResetLastError();
//--- receive chart background color
    if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the background color of chart |
//+-----+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set the chart background color
    if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_COLOR_FOREGROUND** - color of axes, scale and OHLC line.

```

//+-----+
//| Gets the color of axes, scale and OHLC line |
//+-----+
color ChartForeColorGet(const long chart_ID=0)
{
//--- prepare the variable to receive the color
    long result=clrNONE;
//--- reset the error value
    ResetLastError();
//--- receive the color of axes, scale and OHLC line
    if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}

```



```
//+-----+
//| Sets the color of axes, scale and OHLC line |
//+-----+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set the color of axes, scale and OHLC line
    if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_COLOR_GRID** - chart grid color.

```
//+-----+
//| Gets the color of chart grid |
//+-----+
color ChartGridColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
    ResetLastError();
    //--- receive chart grid color
    if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
    }
    //--- return the value of the chart property
    return((color)result);
}

//+-----+
//| Sets the color of chart grid |
//+-----+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set chart grid color
    if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
    {
```



```

    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = "", GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

```

- **CHART_COLOR_VOLUME** - color of volumes and position opening levels.

```

//+-----+
//| Gets the color of volumes and market entry levels |
//+-----+
color ChartVolumeColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
    ResetLastError();
    //--- receive color of volumes and market entry levels
    if(!ChartGetInteger(chart_ID, CHART_COLOR_VOLUME, 0, result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
    }
    //--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the color of volumes and market entry levels |
//+-----+
bool ChartVolumeColorSet(const color clr, const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set color of volumes and market entry levels
    if(!ChartSetInteger(chart_ID, CHART_COLOR_VOLUME, clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```


- **CHART_COLOR_CHART_UP** - color of up bar, its shadow and border of a bullish candlestick's body.

```
//+-----+
//| Gets the color of up bar, shadow and border of a bullish candlestick's body |
//+-----+
color ChartUpColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
    ResetLastError();
    //--- receive the color of up bar, its shadow and border of bullish candlestick's body
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((color)result);
}

//+-----+
//| Sets the color of up bar, shadow and border of a bullish candlestick's body |
//+-----+
bool ChartUpColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set the color of up bar, its shadow and border of body of a bullish candlestick
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_COLOR_CHART_DOWN** - color of down bar, its shadow and border of bearish candlestick's body.

```
//+-----+
//| Gets the color of down bar, shadow and border of a bearish candlestick's body |
//+-----+
color ChartDownColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
```



```

//--- reset the error value
    ResetLastError();
//--- receive the color of down bar, its shadow and border of bearish candlestick's body
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the color of down bar, shadow and border of a bearish candlestick's body |
//+-----+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
//--- set the color of down bar, its shadow and border of bearish candlestick's body
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_COLOR_CHART_LINE** - color of the chart line and Doji candlesticks.

```

//+-----+
//| Gets the color of chart line and Doji candlesticks |
//+-----+
color ChartLineColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
//--- reset the error value
    ResetLastError();
//--- receive color of the chart line and Doji candlesticks
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}

```



```

    }
//+-----+
//| Sets the color of chart line and Doji candlesticks |
//+-----+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set color of the chart line and Doji candlesticks
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

• **CHART_COLOR_CANDLE_BULL** - color of bullish candlestick's body.

```

//+-----+
//| Gets the color of bullish candlestick's body |
//+-----+
color ChartBullColorGet(const long chart_ID=0)
{
//--- prepare the variable to receive the color
    long result=clrNONE;
//--- reset the error value
    ResetLastError();
//--- receive the color of bullish candlestick's body
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the color of bullish candlestick's body |
//+-----+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set the color of bullish candlestick's body
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))

```



```

    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

• **CHART_COLOR_CANDLE_BEAR** - color of bearish candlestick's body.

```

//+-----+
//| Gets the color of bearish candlestick's body |
//+-----+
color ChartBearColorGet(const long chart_ID=0)
{
//--- prepare the variable to receive the color
    long result=clrNONE;
//--- reset the error value
    ResetLastError();
//--- receive the color of bearish candlestick's body
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ", GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the color of bearish candlestick's body |
//+-----+
bool ChartBearColorSet(const color clr,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set the color of bearish candlestick's body
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```


- **CHART_COLOR_BID** - Bid price line color.

```
//+-----+
//| Gets the color of Bid line |
//+-----+
color ChartBidColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
    ResetLastError();
    //--- receive the color of Bid price line
    if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return((color)result);
}

//+-----+
//| Sets the color of Bid line |
//+-----+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set the color of Bid price line
    if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_COLOR_ASK** - Ask price line color.

```
//+-----+
//| Gets the color of Ask line |
//+-----+
color ChartAskColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
```



```

    ResetLastError();
//--- receive the color of Ask price line
    if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}
//+-----+
//| Sets the color of Ask line |
//+-----+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set the color of Ask price line
    if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_COLOR_LAST** - color of the last performed deal's price line (Last).

```

//+-----+
//| Gets the color of the last performed deal's price line |
//+-----+
color ChartLastColorGet(const long chart_ID=0)
{
//--- prepare the variable to receive the color
    long result=clrNONE;
//--- reset the error value
    ResetLastError();
//--- receive color of the last performed deal's price line (Last)
    if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return((color)result);
}

```



```
//+-----+
//| Sets the color of the last performed deal's price line |
//+-----+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set color of the last performed deal's price line (Last)
    if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_COLOR_STOP_LEVEL** - stop order level color (Stop Loss and Take Profit).

```
//+-----+
//| Gets the color of Stop Loss and Take Profit levels |
//+-----+
color ChartStopLevelColorGet(const long chart_ID=0)
{
    //--- prepare the variable to receive the color
    long result=clrNONE;
    //--- reset the error value
    ResetLastError();
    //--- receive the color of stop order levels (Stop Loss and Take Profit)
    if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
    }
    //--- return the value of the chart property
    return((color)result);
}

//+-----+
//| Sets the color of Stop Loss and Take Profit levels |
//+-----+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set the color of stop order levels (Stop Loss and Take Profit)
    if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
    {
```



```

    //--- display the error message in Experts journal
    Print(__FUNCTION__+"", Error Code = "", GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

```

- **CHART_SHOW_TRADE_LEVELS** - property of displaying trade levels on the chart (levels of open positions, Stop Loss, Take Profit and pending orders).

```

//+-----+
//| Checks if trading levels are displayed on chart |
//+-----+
bool ChartShowTradeLevelsGet(bool &result, const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID, CHART_SHOW_TRADE_LEVELS, 0, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of trading levels |
//+-----+
bool ChartShowTradeLevelsSet(const bool value, const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID, CHART_SHOW_TRADE_LEVELS, 0, value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```



```
}
```

- **CHART_DRAG_TRADE_LEVELS** - property of enabling the ability to drag trading levels on a chart using mouse.

```
//+-----+
//| Checks if dragging of trading levels on chart using mouse is allowed |
//+-----+
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
{
    //--- prepare the variable to get the property value
    long value;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}

//+-----+
//| Enables/disables dragging of trading levels on chart using mouse |
//+-----+
bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
```

- **CHART_SHOW_DATE_SCALE** - property of displaying the time scale on a chart.

```
//+-----+
```



```

//| Checks if the time scale is displayed on chart |
//+-----+
bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of the time scale on chart |
//+-----+
bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SHOW_PRICE_SCALE** - property of displaying the price scale on a chart.

```

//+-----+
//| Checks if the price scale is displayed on chart |
//+-----+
bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value

```



```

    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- store the value of the chart property in memory
    result=value;
//--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of the price scale on chart |
//+-----+
bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_SHOW_ONE_CLICK** - property of displaying the "One click trading" panel on a chart.

```

//+-----+
//| Checks if the "One click trading" panel is displayed on chart |
//+-----+
bool ChartShowOneClickPanelGet(bool &result,const long chart_ID=0)
{
//--- prepare the variable to get the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
}

```



```

    }
    //--- store the value of the chart property in memory
    result=value;
    //--- successful execution
    return(true);
}
//+-----+
//| Enables/disables displaying of the "One click trading" panel |
//| on chart |
//+-----+
bool ChartShowOneClickPanelSet(const bool value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_SHIFT_SIZE** - shift size of the zero bar from the right border in percentage values.

```

//+-----+
//| Gets the size of shifting of the zero bar from the right border |
//| of the chart in percentage values (from 10% up to 50%) |
//+-----+
double ChartShiftSizeGet(const long chart_ID=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}
//+-----+
//| Gets the size of shifting of the zero bar from the right border |

```



```

    || of the chart in percentage values (from 10% up to 50%). |
    || To enable the shift mode, CHART_SHIFT property value should be set to true. |
    ||+-----+
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_FIXED_POSITION** - chart fixed position from the left border in percentage value.

```

    ||+-----+
    || Gets the location of chart's fixed position from the left border (in percentage va
    ||+-----+
double ChartFixedPositionGet(const long chart_ID=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}
    ||+-----+
    || Gets the location of chart's fixed position from the left border (in percentage va
    || To view the location of chart's fixed position, the value of CHART_AUTOSCROLL prop
    || should be set to false.
    ||+-----+
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value

```



```

    if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_FIXED_MAX** - property of the chart's fixed maximum.

```

//+-----+
//| Gets the value of chart's fixed maximum |
//+-----+
double ChartFixedMaxGet(const long chart_ID=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}

//+-----+
//| Sets the value of chart's fixed maximum. |
//| To change the value of the property, CHART_SCALEFIX property |
//| value should be preliminarily set to true. |
//+-----+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```



```
}
```

- **CHART_FIXED_MIN** - property of the chart's fixed minimum.

```
//+-----+
//| Gets the value of chart's fixed minimum |
//+-----+
double ChartFixedMinGet(const long chart_ID=0)
{
//--- prepare the variable to get the result
    double result=EMPTY_VALUE;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
//--- return the value of the chart property
    return(result);
}

//+-----+
//| Sets the value of chart's fixed minimum. |
//| To change the value of the property, CHART_SCALEFIX property |
//| value should be preliminarily set to true. |
//+-----+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
```

- **CHART_POINTS_PER_BAR** - value of scale in points per bar.

```
//+-----+
//| Gets the value of chart scale in points per bar |
//+-----+
```



```

double ChartPointsPerBarGet(const long chart_ID=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}

//+-----+
//| Sets the value of chart scale in points per bar. |
//| To view the result of this property's value change, the value of |
//| CHART_SCALE_PT_PER_BAR property should be preliminarily set to true. |
//+-----+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- set property value
    if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```

- **CHART_PRICE_MIN** returns the value of the chart minimum.

```

//+-----+
//| Gets the value of chart minimum in the main window or in a subwindow |
//+-----+
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))

```



```

    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}

```

- **CHART_PRICE_MAX** returns the value of the chart maximum.

```

//+-----+
//| Gets the value of chart maximum in the main window or in a subwindow |
//+-----+
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
{
    //--- prepare the variable to get the result
    double result=EMPTY_VALUE;
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
    }
    //--- return the value of the chart property
    return(result);
}

```

- **CHART_COMMENT** - comment on the chart.

```

//+-----+
//| Gets comment in the upper left corner of chart |
//+-----+
bool ChartCommentGet(string &result,const long chart_ID=0)
{
    //--- reset the error value
    ResetLastError();
    //--- receive the property value
    if(!ChartGetString(chart_ID,CHART_COMMENT,result))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

```



```

    }
//+-----+
//| Gets comment in the upper left corner of chart |
//+-----+
bool ChartCommentSet(const string str,const long chart_ID=0)
{
//--- reset the error value
    ResetLastError();
//--- set property value
    if(!ChartSetString(chart_ID,CHART_COMMENT,str))
    {
        //--- display the error message in Experts journal
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```

- **CHART_IS_MAXIMIZED** - chart window is maximized.

```

//+-----+
//| Defines if the current chart window is maximized |
//+-----+
bool ChartWindowsIsMaximized(bool &result,const long chart_ID=0)
{
//--- prepare the variable for receiving the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_IS_MAXIMIZED))
    {
        //--- display an error message in the Experts log
        Print(__FUNCTION__+" Error Code = ",GetLastError());
        return(false);
    }
//--- store the chart property value in the variable
    result=value;
//--- successful execution
    return(true);
}

```

- **CHART_IS_MINIMIZED** - chart window is minimized.


```

//+-----+
//| Defines if the current chart window is minimized |
//+-----+
bool ChartWindowsIsMinimized(bool &result,const long chart_ID=0)
{
//--- prepare the variable for receiving the property value
    long value;
//--- reset the error value
    ResetLastError();
//--- receive the property value
    if(!ChartGetInteger(chart_ID,CHART_IS_MINIMIZED))
    {
        //--- display an error message in the Experts log
        Print(__FUNCTION__+"", Error Code = "",GetLastError());
        return(false);
    }
//--- store the chart property value in the variable
    result=value;
//--- successful execution
    return(true);
}

```

Panel for chart properties

```

//--- connect the library of control elements
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- predefined constants
#define X_PROPERTY_NAME_1    10  // x coordinate of the property name in the first column
#define X_PROPERTY_VALUE_1  225 // x coordinate of the property value in the first column
#define X_PROPERTY_NAME_2   345 // x coordinate of the property name in the second column
#define X_PROPERTY_VALUE_2  550 // x coordinate of the property value in the second column
#define X_BUTTON_1          285 // x coordinate of the button in the first column
#define X_BUTTON_2          700 // x coordinate of the button in the second column
#define Y_PROPERTY_1        30  // y coordinate of the beginning of the first and second columns
#define Y_PROPERTY_2        286 // y coordinate of the beginning of the third column
#define Y_DISTANCE          16  // y axial distance between the lines
#define LAST_PROPERTY_NUMBER 111 // number of the last graphical property
//--- input parameters
input color InpFirstColor=clrDodgerBlue; // Color of odd lines
input color InpSecondColor=clrGoldenrod; // Color of even lines
//--- variables and arrays
CChartObjectLabel ExtLabelsName[]; // labels for displaying property names
CChartObjectLabel ExtLabelsValue[]; // labels for displaying property values
CChartObjectButton ExtButtons[]; // buttons
int ExtNumbers[]; // property indices
string ExtNames[]; // property names
uchar ExtDataTypes[]; // property data types (integer, double, string)
uint ExtGroupTypes[]; // array that stores the data on belonging of properties to groups
uchar ExtDrawTypes[]; // array that stores the data on the type of property
double ExtMaxValue[]; // maximum property values that are possible when the property is changed
double ExtMinValue[]; // minimum property values that are possible when the property is changed

```



```

double      ExtStep[];           // steps for changing properties
int         ExtCount;           // total number of all properties
color       ExtColors[2];       // array of colors for displaying lines
string      ExtComments[2];     // array of comments (for CHART_COMMENT property)
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- display a comment on the chart
    Comment("SomeComment");
//--- store colors in the array to be able to switch between them later
    ExtColors[0]=InpFirstColor;
    ExtColors[1]=InpSecondColor;
//--- store comments in the array to be able to switch between them later
    ExtComments[0]="FirstComment";
    ExtComments[1]="SecondComment";
//--- prepare and display the control panel for managing chart properties
    if(!PrepareControls())
        return(INIT_FAILED);
//--- successful execution
    return(INIT_SUCCEEDED);
}
//+-----+
//| Deinitialization function of the expert |
//+-----+
void OnDeinit(const int reason)
{
//--- remove the comment on the chart
    Comment("");
}
//+-----+
//| Handler of a chart event |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- check the event of clicking the chart object
    if(id==CHARTEVENT_OBJECT_CLICK)
    {
        //--- divide the object name by separator
        string obj_name[];
        StringSplit(sparam, '_', obj_name);
        //--- check if the object is a button
        if(obj_name[0]=="Button")
        {
            //--- receive button index

```



```

        int index=(int)StringToInteger(obj_name[1]);
        //--- unpress the button
        ExtButtons[index].State(false);
        //--- set the new value of the property depending on its type
        if(ExtDataTypes[index]=='I')
            ChangeIntegerProperty(index);
        if(ExtDataTypes[index]=='D')
            ChangeDoubleProperty(index);
        if(ExtDataTypes[index]=='S')
            ChangeStringProperty(index);
    }
}

//--- re-draw property values
RedrawProperties();
ChartRedraw();
}

//+-----+
//| Changes an integer property of chart |
//+-----+
void ChangeIntegerProperty(const int index)
{
    //--- receive the current property value
    long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index]);
    //--- define the following property value
    switch(ExtDrawTypes[index])
    {
        case 'C':
            value=GetNextColor((color)value);
            break;
        default:
            value=(long)GetNextValue((double)value,index);
            break;
    }
    //--- set the new property value
    ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,value);
}

//+-----+
//| Changes a double property of chart |
//+-----+
void ChangeDoubleProperty(const int index)
{
    //--- receive the current property value
    double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index]);
    //--- define the following property value
    value=GetNextValue(value,index);
    //--- set the new property value
    ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
}

//+-----+

```



```

//| Changes a string property of chart |
//+-----+
void ChangeStringProperty(const int index)
{
//--- static variable for switching inside ExtComments array
    static uint comment_index=1;
//--- change index for receiving another comment
    comment_index=1-comment_index;
//--- set the new property value
    ChartSetString(0, (ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComments[comment_index])
}
//+-----+
//| Gets the next property value |
//+-----+
double GetNextValue(const double value,const int index)
{
    if (value+ExtStep[index]<=ExtMaxValue[index])
        return (value+ExtStep[index]);
    else
        return (ExtMinValue[index]);
}
//+-----+
//| Gets the next color for color type property |
//+-----+
color GetNextColor(const color clr)
{
//--- return the following color value
    switch(clr)
    {
        case clrWhite: return(clrRed);
        case clrRed:   return(clrGreen);
        case clrGreen: return(clrBlue);
        case clrBlue:  return(clrBlack);
        default:       return(clrWhite);
    }
}
//+-----+
//| Re-draws property values |
//+-----+
void RedrawProperties(void)
{
//--- property value text
    string text;
    long   value;
//--- loop of the number of properties
    for(int i=0;i<ExtCount;i++)
    {
        text="";
        switch(ExtDataTypes[i])

```



```

{
    case 'I':
        //--- receive the current property value
        if(!ChartGetInteger(0, (ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[i], 0, value))
            break;
        //--- integer property text
        switch(ExtDrawTypes[i])
        {
            //--- color property
            case 'C':
                text=(string)((color)value);
                break;
            //--- boolean property
            case 'B':
                text=(string)((bool)value);
                break;
            //--- ENUM_CHART_MODE enumeration property
            case 'M':
                text=EnumToString((ENUM_CHART_MODE)value);
                break;
            //--- ENUM_CHART_VOLUME_MODE enumeration property
            case 'V':
                text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                break;
            //--- int type number
            default:
                text=IntegerToString(value);
                break;
        }
        break;
    case 'D':
        //--- double property text
        text=DoubleToString(ChartGetDouble(0, (ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[i], 0, value));
        break;
    case 'S':
        //--- string property text
        text=ChartGetString(0, (ENUM_CHART_PROPERTY_STRING)ExtNumbers[i]);
        break;
}

//--- display property value
ExtLabelsValue[i].Description(text);
}

}

//+-----+
//| Creates panel for managing chart properties |
//+-----+

bool PrepareControls(void)
{
    //--- allocate memory for arrays with a reserve

```



```

MemoryAllocation(LAST_PROPERTY_NUMBER+1);
//--- variables
int i=0;      // loop variable
int col_1=0; // number of properties in the first column
int col_2=0; // number of properties in the second column
int col_3=0; // number of properties in the third column
//--- current number of properties - 0
ExtCount=0;
//--- looking for properties in the loop
while(i<=LAST_PROPERTY_NUMBER)
{
    //--- store the current number of the property
    ExtNumbers[ExtCount]=i;
    //--- increase the value of the loop variable
    i++;
    //--- check if there is a property with such a number
    if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[ExtCount],ExtDataValues[ExtCount])>0)
    {
        //--- create control elements for the property
        switch(ExtGroupTypes[ExtCount])
        {
            case 1:
                //--- create labels and a button for the property
                if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VALUE_1,X_BUTTON_1))
                    return(false);
                //--- number of the elements in the first column has increased
                col_1++;
                break;
            case 2:
                //--- create labels and a button for the property
                if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,X_BUTTON_2))
                    return(false);
                //--- number of the elements in the second column has increased
                col_2++;
                break;
            case 3:
                //--- create only labels for the property
                if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,0,Y_PROPERTY_NAME_3))
                    return(false);
                //--- number of the elements in the third column has increased
                col_3++;
                break;
        }
        //--- define maximum and minimum property value and step
        GetMaxMinStep(ExtNumbers[ExtCount],ExtMaxValue[ExtCount],ExtMinValue[ExtCount],ExtStep[ExtCount]);
        //--- increase the number of properties
        ExtCount++;
    }
}

```



```

//--- free the memory not used by arrays
MemoryAllocation(ExtCount);
//--- re-draw property values
RedrawProperties();
ChartRedraw();
//--- successful execution
return(true);
}
//+-----+
//| Allocates memory for arrays |
//+-----+
void MemoryAllocation(const int size)
{
    ArrayResize(ExtLabelsName,size);
    ArrayResize(ExtLabelsValue,size);
    ArrayResize(ExtButtons,size);
    ArrayResize(ExtNumbers,size);
    ArrayResize(ExtNames,size);
    ArrayResize(ExtDataTypes,size);
    ArrayResize(ExtGroupTypes,size);
    ArrayResize(ExtDrawTypes,size);
    ArrayResize(ExtMaxValue,size);
    ArrayResize(ExtMinValue,size);
    ArrayResize(ExtStep,size);
}
//+-----+
//| Checks if the property index belongs to the one of |
//| ENUM_CHART_PROPERTIES enumerations |
//+-----+
bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_type,uchar &
{
    //--- check if the property is of integer type
    ResetLastError();
    name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
    if(_LastError==0)
    {
        data_type='I'; // property from ENUM_CHART_PROPERTY_INTEGER
        GetTypes(ind,group_type,draw_type); // define property display parameters
        return(true);
    }
    //--- check if the property is of double type
    ResetLastError();
    name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
    if(_LastError==0)
    {
        data_type='D'; // property from ENUM_CHART_PROPERTY_DOUBLE
        GetTypes(ind,group_type,draw_type); // define property display parameters
        return(true);
    }
}

```



```

//--- check if the property is of string type
ResetLastError();
name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
if(_LastError==0)
{
    data_type='S'; // property from ENUM_CHART_PROPERTY_STRING
    GetTypes(ind,group_type,draw_type); // define property display parameters
    return(true);
}
//--- property does not belong to any enumeration
return(false);
}
//+-----+
//| Defines the group in which property should be stored, |
//| as well as its display type |
//+-----+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
{
    //--- check if the property belongs to the third group
    //--- third group properties are displayed in the second column starting from CHART_B
    if(CheckThirdGroup(property_number,group_type,draw_type))
        return;
    //--- check if the property belongs to the second group
    //--- second group properties are displayed at the beginning of the second column
    if(CheckSecondGroup(property_number,group_type,draw_type))
        return;
    //--- if you find yourself here, the property belongs to the first group (first column)
    CheckFirstGroup(property_number,group_type,draw_type);
}
//+-----+
//| Checks if property belongs to the third group and |
//| defines its display type in case of a positive answer |
//+-----+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &draw_type)
{
    //--- check if the property belongs to the third group
    switch(property_number)
    {
        //--- boolean properties
        case CHART_IS_OBJECT:
        case CHART_WINDOW_IS_VISIBLE:
            draw_type='B';
            break;
        //--- integer properties
        case CHART_VISIBLE_BARS:
        case CHART_WINDOWS_TOTAL:
        case CHART_WINDOW_HANDLE:
        case CHART_WINDOW_YDISTANCE:
        case CHART_FIRST_VISIBLE_BAR:

```



```

    case CHART_WIDTH_IN_BARS:
    case CHART_WIDTH_IN_PIXELS:
        draw_type='I';
        break;
        //--- double properties
    case CHART_PRICE_MIN:
    case CHART_PRICE_MAX:
        draw_type='D';
        break;
        //--- in fact, this property is a command of displaying the chart on top of a
        //--- there is no need to apply this panel, as the window will always be
        //--- on top of other ones before we use it
    case CHART_BRING_TO_TOP:
        draw_type=' ';
        break;
        //--- property does not belong to the third group
    default:
        return(false);
    }
    //--- property belongs to the third group
    group_type=3;
    return(true);
}
//+-----+
//| Checks if property belongs to the second group and |
//| defines its display type in case of a positive answer |
//+-----+
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &draw_type)
{
    //--- check if the property belongs to the second group
    switch(property_number)
    {
        //--- ENUM_CHART_MODE type property
    case CHART_MODE:
        draw_type='M';
        break;
        //--- ENUM_CHART_VOLUME_MODE type property
    case CHART_SHOW_VOLUMES:
        draw_type='V';
        break;
        //--- string property
    case CHART_COMMENT:
        draw_type='S';
        break;
        //--- color property
    case CHART_COLOR_BACKGROUND:
    case CHART_COLOR_FOREGROUND:
    case CHART_COLOR_GRID:
    case CHART_COLOR_VOLUME:

```



```

        case CHART_COLOR_CHART_UP:
        case CHART_COLOR_CHART_DOWN:
        case CHART_COLOR_CHART_LINE:
        case CHART_COLOR_CANDLE_BULL:
        case CHART_COLOR_CANDLE_BEAR:
        case CHART_COLOR_BID:
        case CHART_COLOR_ASK:
        case CHART_COLOR_LAST:
        case CHART_COLOR_STOP_LEVEL:
            draw_type='C';
            break;
        //--- property does not belong to the second group
    default:
        return(false);
    }
//--- property belongs to the second group
    group_type=2;
    return(true);
}
//+-----+
//| Called only if it is already known that property does not belong      |
//| to the second and third property groups                                |
//+-----+
void CheckFirstGroup(const int property_number,uint &group_type,uchar &draw_type)
{
//--- the property belongs to the first group
    group_type=1;
//--- define property display type
    switch(property_number)
    {
        //--- integer properties
        case CHART_SCALE:
        case CHART_HEIGHT_IN_PIXELS:
            draw_type='I';
            return;
        //--- double properties
        case CHART_SHIFT_SIZE:
        case CHART_FIXED_POSITION:
        case CHART_FIXED_MAX:
        case CHART_FIXED_MIN:
        case CHART_POINTS_PER_BAR:
            draw_type='D';
            return;
        //--- only boolean properties have remained
    default:
        draw_type='B';
        return;
    }
}

```



```

//+-----+
//| Creates label and button for property |
//+-----+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
{
//--- static array for switching inside ExtColors color array
    static uint color_index[3]={1,1,1};
//--- change index for receiving another color
    color_index[type]=1-color_index[type];
//--- display labels and a button (if btn=true) for the property
    if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],ExtColors[color_index[t
        return(false);
    if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[color_index[t
        return(false);
    if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
        return(false);
//--- successful execution
    return(true);
}
//+-----+
//| Creates label |
//+-----+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string text,
                 const color clr,const int x,const int y)
{
    if(!lbl.Create(0,"Label_"+name,0,x,y)) return(false);
    if(!lbl.Description(text)) return(false);
    if(!lbl.FontSize(10)) return(false);
    if(!lbl.Color(clr)) return(false);
//--- successful execution
    return(true);
}
//+-----+
//| Creates button |
//+-----+
bool ButtonCreate(CChartObjectButton &btn,const string name,
                 const int x,const int y)
{
    if(!btn.Create(0,"Button_"+name,0,x,y,50,15)) return(false);
    if(!btn.Description("Next")) return(false);
    if(!btn.FontSize(10)) return(false);
    if(!btn.Color(clrBlack)) return(false);
    if(!btn.BackColor(clrWhite)) return(false);
    if(!btn.BorderColor(clrBlack)) return(false);
//--- successful execution
    return(true);
}
//+-----+

```



```

///| Defines maximum and minimum property value and step |
//+-----+
void GetMaxMinStep(const int property_number, double &max, double &min, double &step)
{
    double value;
    ///--- set values depending on the property type
    switch(property_number)
    {
        case CHART_SCALE:
            max=5;
            min=0;
            step=1;
            break;
        case CHART_MODE:
        case CHART_SHOW_VOLUMES:
            max=2;
            min=0;
            step=1;
            break;
        case CHART_SHIFT_SIZE:
            max=50;
            min=10;
            step=2.5;
            break;
        case CHART_FIXED_POSITION:
            max=90;
            min=0;
            step=15;
            break;
        case CHART_POINTS_PER_BAR:
            max=19;
            min=1;
            step=3;
            break;
        case CHART_FIXED_MAX:
            value=ChartGetDouble(0, CHART_FIXED_MAX);
            max=value*1.25;
            min=value;
            step=value/32;
            break;
        case CHART_FIXED_MIN:
            value=ChartGetDouble(0, CHART_FIXED_MIN);
            max=value;
            min=value*0.75;
            step=value/32;
            break;
        case CHART_HEIGHT_IN_PIXELS:
            max=700;
            min=520;
    }
}

```



```
    step=30;
    break;
    ///--- default values
default:
    max=1;
    min=0;
    step=1;
}
}
```


Object Constants

There are 44 graphical objects that can be created and displayed in the price chart. All constants for working with objects are divided into 9 groups:



















- [Object types](#) - Identifiers of graphical objects;
- [Object properties](#) - setting and getting properties of graphical objects;
- [Methods of object binding](#) - constants of object positioning in the chart;
- [Binding corner](#) - setting the corner relative to which an object is positioned on chart;
- [Visibility of objects](#) - setting timeframes in which an object is visible;
- [Levels of Elliott Waves](#) - gradation of waves;
- [Gann objects](#) - trend constants for Gann fan and Gann grid;
- [Web colors](#) - constants of predefined web colors;
- [Wingdings](#) - codes of characters of the Wingdings font.

Object Types

When a graphical object is created using the [ObjectCreate\(\)](#) function, it's necessary to specify the type of object being created, which can be one of the values of the ENUM_OBJECT enumeration. Further specifications of object [properties](#) are possible using functions for working with [graphical objects](#).

ENUM_OBJECT

ID		Description
OBJ_VLINE		Vertical Line
OBJ_HLINE	—	Horizontal Line
OBJ_TREND	/	Trend Line
OBJ_TRENDBYANGLE		Trend Line By Angle
OBJ_CYCLES		Cycle Lines
OBJ_ARROWED_LINE		Arrowed Line
OBJ_CHANNEL		Equidistant Channel
OBJ_STDDEVCHANNEL		Standard Deviation Channel
OBJ_REGRESSION		Linear Regression Channel
OBJ_PITCHFORK		Andrews' Pitchfork
OBJ_GANNLIN		Gann Line
OBJ_GANNFAN		Gann Fan
OBJ_GANNGRID		Gann Grid
OBJ_FIBO		Fibonacci Retracement
OBJ_FIBOTIMES		Fibonacci Time Zones
OBJ_FIBOFAN		Fibonacci Fan
OBJ_FIBOARC		Fibonacci Arcs
OBJ_FIBOCHANNEL		Fibonacci Channel
OBJ_EXPANSION		Fibonacci Expansion
OBJ_ELLIOTWAVE5		Elliott Motive Wave
OBJ_ELLIOTWAVE3		Elliott Correction Wave
OBJ_RECTANGLE		Rectangle
OBJ_TRIANGLE		Triangle
OBJ_ELLIPSE		Ellipse
OBJ_ARROW_THUMB_UP		Thumbs Up
OBJ_ARROW_THUMB_DOWN		Thumbs Down

<u>OBJ_ARROW_UP</u>		Arrow Up
<u>OBJ_ARROW_DOWN</u>		Arrow Down
<u>OBJ_ARROW_STOP</u>		Stop Sign
<u>OBJ_ARROW_CHECK</u>		Check Sign
<u>OBJ_ARROW_LEFT_PRICE</u>		Left Price Label
<u>OBJ_ARROW_RIGHT_PRICE</u>		Right Price Label
<u>OBJ_ARROW_BUY</u>		Buy Sign
<u>OBJ_ARROW_SELL</u>		Sell Sign
<u>OBJ_ARROW</u>		Arrow
<u>OBJ_TEXT</u>		Text
<u>OBJ_LABEL</u>		Label
<u>OBJ_BUTTON</u>		Button
<u>OBJ_CHART</u>		Chart
<u>OBJ_BITMAP</u>		Bitmap
<u>OBJ_BITMAP_LABEL</u>		Bitmap Label
<u>OBJ_EDIT</u>		Edit
<u>OBJ_EVENT</u>		The "Event" object corresponding to an event in the economic calendar
<u>OBJ_RECTANGLE_LABEL</u>		The "Rectangle label" object for creating and designing the custom graphical interface.

OBJ_VLINE

Vertical Line.



Note

When drawing a vertical line, it is possible to set the line display mode for all chart windows (property [OBJPROP_RAY](#)).

Example

The following script creates and moves the vertical line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Vertical Line\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="VLine";      // Line name
input int         InpDate=25;           // Event date, %
input color       InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int         InpWidth=3;           // Line width
input bool        InpBack=false;        // Background line
input bool        InpSelection=true;    // Highlight to move
input bool        InpRay=true;          // Line's continuation down
```



```

input bool      InpHidden=true;      // Hidden in the object list
input long      InpZOrder=0;         // Priority for mouse click
//+-----+
//| Create the vertical line          |
//+-----+
bool VLineCreate(const long          chart_ID=0,      // chart's ID
                 const string       name="VLine",    // line name
                 const int          sub_window=0,    // subwindow index
                 datetime            time=0,         // line time
                 const color        clr=clrRed,      // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int          width=1,        // line width
                 const bool         back=false,     // in the background
                 const bool         selection=true,  // highlight to move
                 const bool         ray=true,       // line's continuation down
                 const bool         hidden=true,    // hidden in the object list
                 const long          z_order=0)      // priority for mouse click
{
//--- if the line time is not set, draw it via the last bar
    if(!time)
        time=TimeCurrent();
//--- reset the error value
    ResetLastError();
//--- create a vertical line
    if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
    {
        Print(__FUNCTION__,
              ": failed to create a vertical line! Error code = ",GetLastError());
        return(false);
    }
//--- set line color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of displaying the line in the chart s
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the vertical line |
//+-----+
bool VLineMove(const long   chart_ID=0,    // chart's ID
               const string name="VLine", // line name
               datetime     time=0)        // line time
{
//--- if line time is not set, move the line to the last bar
    if(!time)
        time=TimeCurrent();
//--- reset the error value
    ResetLastError();
//--- move the vertical line
    if(!ObjectMove(chart_ID,name,0,time,0))
    {
        Print(__FUNCTION__,
              ": failed to move the vertical line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete the vertical line |
//+-----+
bool VLineDelete(const long   chart_ID=0,    // chart's ID
                 const string name="VLine") // line name
{
//--- reset the error value
    ResetLastError();
//--- delete the vertical line
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete the vertical line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

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

```



```

//--- check correctness of the input parameters
if(InpDate<0 || InpDate>100)
{
    Print("Error! Incorrect values of input parameters!");
    return;
}

//--- number of visible bars in the chart window
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing line anchor point's coordinates
datetime date[];
//--- memory allocation
ArrayResize(date,bars);
//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}

//--- define points for drawing the line
int d=InpDate*(bars-1)/100;
//--- create a vertical line
if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
    InpSelection,InpRay,InpHidden,InpZOrder))
    return;

//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);

//--- now, move the line
//--- loop counter
int h_steps=bars/2;
//--- move the line
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d<bars-1)
        d+=1;
    //--- move the point
    if(!VLineMove(0,InpName,date[d]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.03 seconds of delay
    Sleep(30);
}

```



```
//--- 1 second of delay
    Sleep(1000);
//--- delete the channel from the chart
    VLineDelete(0, InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---
}
```


OBJ_HLINE

Horizontal Line.



Example

The following script creates and moves the horizontal line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Horizontal Line\" graphical object."
#property description "Anchor point price is set in percentage of the height of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="HLine";      // Line name
input int         InpPrice=25;           // Line price, %
input color       InpColor=clrRed;       // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int         InpWidth=3;            // Line width
input bool        InpBack=false;         // Background line
input bool        InpSelection=true;     // Highlight to move
input bool        InpHidden=true;        // Hidden in the object list
input long        InpZOrder=0;           // Priority for mouse click
//+-----+
//| Create the horizontal line |
//+-----+
```



```

bool HLineCreate(const long      chart_ID=0,      // chart's ID
                 const string   name="HLine",    // line name
                 const int      sub_window=0,    // subwindow index
                 double         price=0,         // line price
                 const color     clr=clrRed,     // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int      width=1,        // line width
                 const bool     back=false,     // in the background
                 const bool     selection=true,  // highlight to move
                 const bool     hidden=true,    // hidden in the object list
                 const long      z_order=0)      // priority for mouse click
{
//--- if the price is not set, set it at the current Bid price level
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- create a horizontal line
if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
{
    Print(__FUNCTION__,
          ": failed to create a horizontal line! Error code = ",GetLastError());
    return(false);
}
//--- set line color
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}
//+-----+
//| Move horizontal line |
//+-----+
bool HLineMove(const long      chart_ID=0,      // chart's ID

```



```

        const string name="HLine", // line name
        double      price=0)      // line price
    {
//--- if the line price is not set, move it to the current Bid price level
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move a horizontal line
    if(!ObjectMove(chart_ID,name,0,0,price))
    {
        Print(__FUNCTION__,
            ": failed to move the horizontal line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete a horizontal line |
//+-----+
bool HLineDelete(const long   chart_ID=0, // chart's ID
                 const string name="HLine") // line name
{
//--- reset the error value
    ResetLastError();
//--- delete a horizontal line
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete a horizontal line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- price array size
    int accuracy=1000;

```



```

//--- array for storing the price values to be used
//--- for setting and changing line anchor point's coordinates
    double price[];
//--- memory allocation
    ArrayResize(price,accuracy);
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the line
    int p=InpPrice*(accuracy-1)/100;
//--- create a horizontal line
    if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack,
        InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the line
//--- loop counter
    int v_steps=accuracy/2;
//--- move the line
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p<accuracy-1)
            p+=1;
        //--- move the point
        if(!HLineMove(0,InpName,price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
//--- 1 second of delay
    Sleep(1000);
//--- delete from the chart
    HLineDelete(0,InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);

```



```
//---  
}
```


OBJ_TREND

Trend Line.



Note

For Trend Line, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Trend Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Trend";      // Line name
input int         InpDate1=35;          // 1 st point's date, %
input int         InpPrice1=60;         // 1 st point's price, %
input int         InpDate2=65;         // 2 nd point's date, %
input int         InpPrice2=40;         // 2 nd point's price, %
input color       InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
```



```

input int      InpWidth=2;          // Line width
input bool     InpBack=false;       // Background line
input bool     InpSelection=true;   // Highlight to move
input bool     InpRayLeft=false;    // Line's continuation to the left
input bool     InpRayRight=false;   // Line's continuation to the right
input bool     InpHidden=true;      // Hidden in the object list
input long     InpZOrder=0;         // Priority for mouse click
//+-----+
//| Create a trend line by the given coordinates |
//+-----+
bool TrendCreate(const long      chart_ID=0,          // chart's ID
                 const string    name="TrendLine",   // line name
                 const int       sub_window=0,        // subwindow index
                 datetime        time1=0,             // first point time
                 double          price1=0,            // first point price
                 datetime        time2=0,             // second point time
                 double          price2=0,            // second point price
                 const color      clr=clrRed,         // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int       width=1,            // line width
                 const bool      back=false,         // in the background
                 const bool      selection=true,      // highlight to move
                 const bool      ray_left=false,     // line's continuation to the left
                 const bool      ray_right=false,    // line's continuation to the right
                 const bool      hidden=true,        // hidden in the object list
                 const long      z_order=0)           // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeTrendEmptyPoints(time1,price1,time2,price2);
    //--- reset the error value
    ResetLastError();
    //--- create a trend line by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create a trend line! Error code = ",GetLastError());
        return(false);
    }
    //--- set line color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set line display style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the line by mouse
    //--- when creating a graphical object using ObjectCreate function, the object cannot
    //--- highlighted and moved by default. Inside this method, selection parameter

```



```

//--- is true by default making it possible to highlight and move the object
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}

//+-----+
//| Move trend line anchor point |
//+-----+
bool TrendPointChange(const long   chart_ID=0,      // chart's ID
                      const string name="TrendLine", // line name
                      const int    point_index=0,   // anchor point index
                      datetime      time=0,         // anchor point time coordinate
                      double        price=0)        // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move trend line's anchor point
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| The function deletes the trend line from the chart. |
//+-----+
bool TrendDelete(const long   chart_ID=0,      // chart's ID
                  const string name="TrendLine") // line name
{
//--- reset the error value
ResetLastError();
//--- delete a trend line

```



```

    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check the values of trend line's anchor points and set default |
//| values for empty ones                                         |
//+-----+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
//--- if the second point's price is not set, it is equal to the first point's one
    if(!price2)
        price2=price1;
}

//+-----+
//| Script program start function                                |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

```



```

//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the line
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
    if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,InpStyle,
        InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the line's anchor points
//--- loop counter
    int v_steps=accuracy/5;
//--- move the first anchor point vertically
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
        if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))

```



```

        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- move the second anchor point vertically
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p2<accuracy-1)
        p2+=1;
    //--- move the point
    if(!TrendPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- half a second of delay
Sleep(500);
//--- loop counter
int h_steps=bars/2;
//--- move both anchor points horizontally at the same time
for(int i=0;i<h_steps;i++)
{
    //--- use the following values
    if(d1<bars-1)
        d1+=1;
    if(d2>1)
        d2-=1;
    //--- shift the points
    if(!TrendPointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    if(!TrendPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.03 seconds of delay
    Sleep(30);
}
//--- 1 second of delay
Sleep(1000);

```



```
//--- delete a trend line
    TrendDelete(0, InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---
}
```


OBJ_TRENDBYANGLE

Trend Line By Angle.



Note

For Trend Line By Angle, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

Both angle and the second anchor point's coordinates can be used to set the slope of the line.

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Trend Line By Angle\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Trend";      // Line name
input int         InpDate1=50;          // 1 st point's date, %
input int         InpPrice1=75;         // 1 st point's price, %
input int         InpAngle=0;           // Line's slope angle
input color       InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
```



```

input int      InpWidth=2;          // Line width
input bool     InpBack=false;       // Background line
input bool     InpSelection=true;    // Highlight to move
input bool     InpRayLeft=false;    // Line's continuation to the left
input bool     InpRayRight=true;    // Line's continuation to the right
input bool     InpHidden=true;      // Hidden in the object list
input long     InpZOrder=0;         // Priority for mouse click
//+-----+
//| Create a trend line by angle |
//+-----+
bool TrendByAngleCreate(const long      chart_ID=0,          // chart's ID
                        const string    name="TrendLine",    // line name
                        const int       sub_window=0,        // subwindow index
                        datetime        time=0,              // point time
                        double          price=0,             // point price
                        const double    angle=45.0,          // slope angle
                        const color     clr=clrRed,          // line color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                        const int       width=1,             // line width
                        const bool      back=false,          // in the background
                        const bool      selection=true,       // highlight to move
                        const bool      ray_left=false,       // line's continuation to the left
                        const bool      ray_right=true,       // line's continuation to the right
                        const bool      hidden=true,          // hidden in the object list
                        const long      z_order=0)           // priority for mouse click
{
    //--- create the second point to facilitate dragging the trend line by mouse
    datetime time2=0;
    double price2=0;
    //--- set anchor points' coordinates if they are not set
    ChangeTrendEmptyPoints(time,price,time2,price2);
    //--- reset the error value
    ResetLastError();
    //--- create a trend line using 2 points
    if(!ObjectCreate(chart_ID,name,OBJ_TRENDBYANGLE,sub_window,time,price,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create a trend line! Error code = ",GetLastError());
        return(false);
    }
    //--- change trend line's slope angle; when changing the angle, coordinates of the second
    //--- point of the line are redefined automatically according to the angle's new value
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
    //--- set line color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
}

```



```

//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Change trend line anchor point's coordinates |
//+-----+
bool TrendPointChange(const long   chart_ID=0,      // chart's ID
                     const string name="TrendLine", // line name
                     datetime     time=0,          // anchor point time coordinate
                     double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move trend line's anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change trend line's slope angle |
//+-----+
bool TrendAngleChange(const long   chart_ID=0,      // chart's ID
                     const string name="TrendLine", // trend line name

```



```

        const double angle=45)           // trend line's slope angle
    {
//--- reset the error value
        ResetLastError();
//--- change trend line's slope angle
        if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
        {
            Print(__FUNCTION__,
                  ": failed to change the line's slope angle! Error code = ",GetLastError());
            return(false);
        }
//--- successful execution
        return(true);
    }
//+-----+
//| Delete the trend line                               |
//+-----+
bool TrendDelete(const long   chart_ID=0,           // chart's ID
                 const string name="TrendLine") // line name
{
//--- reset the error value
        ResetLastError();
//--- delete a trend line
        if(!ObjectDelete(chart_ID,name))
        {
            Print(__FUNCTION__,
                  ": failed to delete a trend line! Error code = ",GetLastError());
            return(false);
        }
//--- successful execution
        return(true);
    }
//+-----+
//| Check the values of trend line's anchor points and set default |
//| values for empty ones                                         |
//+-----+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                           datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
        if(!time1)
            time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
        if(!price1)
            price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- set coordinates of the second, auxiliary point
//--- the second point will be 9 bars left and have the same price
        datetime second_point_time[10];
        CopyTime(Symbol(),Period(),time1,10,second_point_time);

```



```

    time2=second_point_time[0];
    price2=price1;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing line anchor points' coordinates
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- define points for drawing the line
    int d1=InpDate1*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    //--- create a trend line
    if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor,InpStyle,
        InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
    {
        return;
    }
}

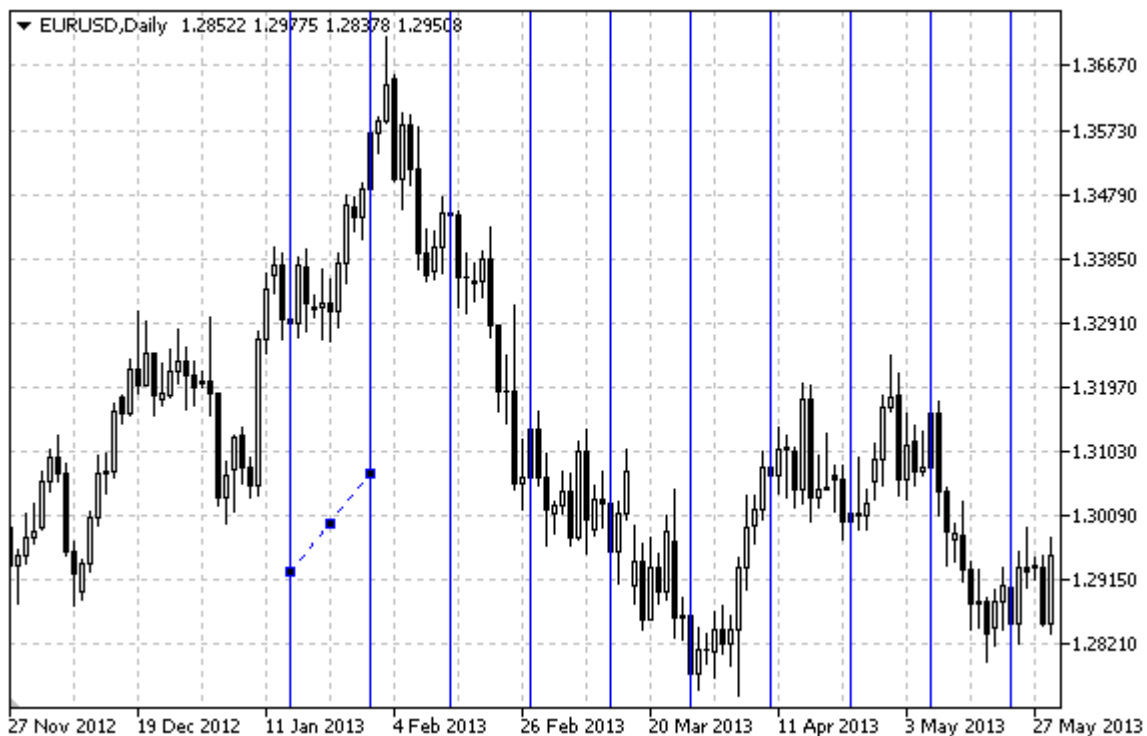
```



```
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move and rotate the line
//--- loop counter
    int v_steps=accuracy/2;
//--- move the anchor point and change the line's slope angle
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
        if(!TrendPointChange(0,InpName,date[d1],price[p1]))
            return;
        if(!TrendAngleChange(0,InpName,18*(i+1)))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
//--- 1 second of delay
    Sleep(1000);
//--- delete from the chart
    TrendDelete(0,InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---
}
```


OBJ_CYCLES

Cycle Lines.



Note

The distance between the lines is set by time coordinates of two anchor points of the object.

Example

The following script creates and moves cycle lines on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates cycle lines on the chart."
#property description "Anchor point coordinates are set in percentage"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Cycles";    // Object name
input int         InpDate1=10;         // 1 st point's date, %
input int         InpPrice1=45;        // 1 st point's price, %
input int         InpDate2=20;         // 2 nd point's date, %
input int         InpPrice2=55;        // 2 nd point's price, %
input color       InpColor=clrRed;     // Color of cycle lines
input ENUM_LINE_STYLE InpStyle=STYLE_DOT; // Style of cycle lines
input int         InpWidth=1;          // Width of cycle lines
```



```

input bool      InpBack=false;      // Background object
input bool      InpSelection=true;   // Highlight to move
input bool      InpHidden=true;     // Hidden in the object list
input long      InpZOrder=0;        // Priority for mouse click
//+-----+
//| Create cycle lines                                     |
//+-----+
bool CyclesCreate(const long          chart_ID=0,      // chart's ID
                  const string        name="Cycles",  // object name
                  const int           sub_window=0,   // subwindow index
                  datetime             time1=0,       // first point time
                  double               price1=0,      // first point price
                  datetime             time2=0,       // second point time
                  double               price2=0,      // second point price
                  const color          clr=clrRed,    // color of cycle lines
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // style of cycle lines
                  const int            width=1,       // width of cycle lines
                  const bool           back=false,    // in the background
                  const bool           selection=true, // highlight to move
                  const bool           hidden=true,   // hidden in the object list
                  const long           z_order=0)     // priority for mouse click
{
//--- set anchor points' coordinates if they are not set
    ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
    ResetLastError();
//--- create cycle lines by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create cycle lines! Error code = ",GetLastError());
        return(false);
    }
//--- set color of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the lines by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

```



```

//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}
//+-----+
//| Move the anchor point |
//+-----+
bool CyclesPointChange(const long   chart_ID=0,    // chart's ID
                      const string name="Cycles", // object name
                      const int    point_index=0, // anchor point index
                      datetime      time=0,       // anchor point time coordinate
                      double        price=0)      // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}
//+-----+
//| Delete the cycle lines |
//+-----+
bool CyclesDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Cycles") // object name
{
//--- reset the error value
ResetLastError();
//--- delete cycle lines
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": failed to delete cycle lines! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

```



```

//+-----+
//| Check the values of cycle lines' anchor points and set default |
//| values for empty ones |
//+-----+
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
//--- if the second point's price is not set, it is equal to the first point's one
    if(!price2)
        price2=price1;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of cycle lines' anchor points
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
}

```



```

//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of prices
//--- find the highest and lowest values of the chart
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing cycle lines
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the anchor points
//--- loop counter
int h_steps=bars/5;
//--- move the second anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d2<bars-1)
        d2+=1;
    //--- move the point
    if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}

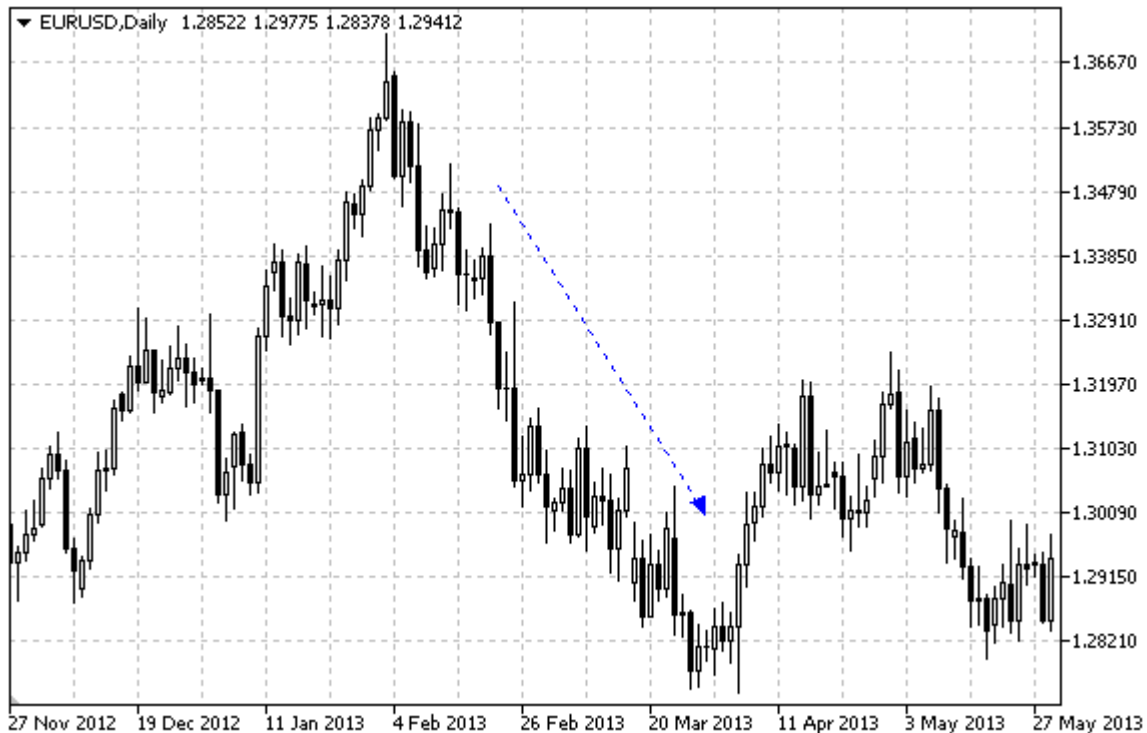
```



```
//--- 1 second of delay
Sleep(1000);
//--- loop counter
h_steps=bars/4;
//--- move the first anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d1<bars-1)
        d1+=1;
    //--- move the point
    if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- delete the object from the chart
CyclesDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_ARROWED_LINE

Arrowed line.



Example

The following script creates and moves an arrow line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Arrowed line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ArrowedLine"; // Line name
input int         InpDate1=35;           // 1 st point's date, %
input int         InpPrice1=60;           // 1 st point's price, %
input int         InpDate2=65;           // 2 nd point's date, %
input int         InpPrice2=40;           // 2 nd point's price, %
input color       InpColor=clrRed;        // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int         InpWidth=2;             // Line width
input bool        InpBack=false;          // Background line
input bool        InpSelection=true;       // Highlight to move
input bool        InpHidden=true;         // Hidden in the object list
input long        InpZOrder=0;            // Priority for mouse click
```



```

//+-----+
//| Create an arrowed line by the given coordinates |
//+-----+
bool ArrowedLineCreate(const long      chart_ID=0,      // chart's ID
                      const string    name="ArrowedLine", // line name
                      const int       sub_window=0,     // subwindow index
                      datetime        time1=0,          // first point time
                      double           price1=0,         // first point price
                      datetime        time2=0,          // second point time
                      double           price2=0,         // second point price
                      const color      clr=clrRed,       // line color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                      const int       width=1,          // line width
                      const bool      back=false,       // in the background
                      const bool      selection=true,   // highlight to move
                      const bool      hidden=true,      // hidden in the object list
                      const long      z_order=0)         // priority for mouse click

{
    //--- set anchor points' coordinates if they are not set
    ChangeArrowedLineEmptyPoints(time1,price1,time2,price2);
    //--- reset the error value
    ResetLastError();
    //--- create an arrowed line by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_ARROWED_LINE,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create an arrowed line! Error code = ",GetLastError());
        return(false);
    }
    //--- set line color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set line display style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the line by mouse
    //--- when creating a graphical object using ObjectCreate function, the object cannot
    //--- be highlighted and moved by default. Inside this method, selection parameter
    //--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
    //--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
    //--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
    //--- successful execution
    return(true);
}

```



```

    }
//+-----+
//| Move arrowed line's anchor point |
//+-----+
bool ArrowedLinePointChange(const long   chart_ID=0,           // chart's ID
                           const string name="ArrowedLine",    // line name
                           const int    point_index=0,         // anchor point index
                           datetime     time=0,               // anchor point time coordinate
                           double       price=0)               // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the line's anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| The function removes the arrowed line from the chart |
//+-----+
bool ArrowedLineDelete(const long   chart_ID=0,           // chart's ID
                      const string name="ArrowedLine")    // line name
{
//--- reset the error value
    ResetLastError();
//--- delete an arrowed line
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to create an arrowed line! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor points' values and set default values |
//| for empty ones |
//+-----+

```



```

void ChangeArrowedLineEmptyPoints(datetime &time1,double &price1,
                                   datetime &time2,double &price2)
{
    //--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
    //--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
    //--- if the second point's price is not set, it is equal to the first point's one
    if(!price2)
        price2=price1;
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing line anchor points' coordinates
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {

```



```

        Print("Failed to copy time values! Error code = ", GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0, CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0, CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0; i<accuracy; i++)
        price[i]=min_price+i*step;
//--- define points for drawing the line
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
//--- create an arrowed line
    if(!ArrowedLineCreate(0, InpName, 0, date[d1], price[p1], date[d2], price[p2],
        InpColor, InpStyle, InpWidth, InpBack, InpSelection, InpHidden, InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the line's anchor points
//--- loop counter
    int v_steps=accuracy/5;
//--- move the second anchor point vertically
    for(int i=0; i<v_steps; i++)
    {
        //--- use the following value
        if(p2<accuracy-1)
            p2+=1;
        //--- move the point
        if(!ArrowedLinePointChange(0, InpName, 1, date[d2], price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
//--- move the first anchor point vertically
    for(int i=0; i<v_steps; i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;

```



```

    //--- move the point
    if(!ArrowedLinePointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- half a second of delay
Sleep(500);
//--- loop counter
int h_steps=bars/2;
//--- move both anchor points horizontally at the same time
for(int i=0; i<h_steps; i++)
{
    //--- use the following values
    if(d1<bars-1)
        d1+=1;
    if(d2>1)
        d2-=1;
    //--- shift the points
    if(!ArrowedLinePointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    if(!ArrowedLinePointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.03 seconds of delay
    Sleep(30);
}
//--- 1 second of delay
Sleep(1000);
//--- delete an arrowed line
ArrowedLineDelete(0, InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}

```


OBJ_CHANNEL

Equidistant Channel



Note

For an equidistant channel, it is possible to specify the mode of its continuation to the right and/or to the left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly). The mode of filling the channel with color can also be set.

Example

The following script creates and moves an equidistant channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Equidistant Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Channel";    // Channel name
input int         InpDate1=25;          // 1 st point's date, %
input int         InpPrice1=60;         // 1 st point's price, %
input int         InpDate2=65;          // 2 nd point's date, %
input int         InpPrice2=80;         // 2 nd point's price, %
input int         InpDate3=30;          // 3 rd point's date, %
```



```

input int      InpPrice3=40;           // 3 rd point's price, %
input color    InpColor=clrRed;        // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of channel lines
input int      InpWidth=2;             // Channel line width
input bool     InpBack=false;          // Background channel
input bool     InpFill=false;          // Filling the channel with color
input bool     InpSelection=true;      // Highlight to move
input bool     InpRayLeft=false;       // Channel's continuation to the left
input bool     InpRayRight=false;      // Channel's continuation to the right
input bool     InpHidden=true;         // Hidden in the object list
input long     InpZOrder=0;            // Priority for mouse click

//+-----+
//| Create an equidistant channel by the given coordinates |
//+-----+

bool ChannelCreate(const long      chart_ID=0,           // chart's ID
                  const string    name="Channel",       // channel name
                  const int       sub_window=0,         // subwindow index
                  datetime        time1=0,              // first point time
                  double          price1=0,             // first point price
                  datetime        time2=0,              // second point time
                  double          price2=0,             // second point price
                  datetime        time3=0,              // third point time
                  double          price3=0,             // third point price
                  const color      clr=clrRed,          // channel color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel lines
                  const int       width=1,             // width of channel lines
                  const bool      fill=false,          // filling the channel with
                  const bool      back=false,          // in the background
                  const bool      selection=true,       // highlight to move
                  const bool      ray_left=false,      // channel's continuation
                  const bool      ray_right=false,     // channel's continuation
                  const bool      hidden=true,         // hidden in the object list
                  const long      z_order=0)           // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- create a channel by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create an equidistant channel! Error code = ",GetLastError());
        return(false);
    }
    //--- set channel color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
}

```



```

//--- set width of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the channel
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the channel's anchor point |
//+-----+
bool ChannelPointChange(const long   chart_ID=0,    // chart's ID
                        const string name="Channel", // channel name
                        const int    point_index=0, // anchor point index
                        datetime      time=0,       // anchor point time coordinate
                        double        price=0)       // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```



```

//+-----+
//| Delete the channel |
//+-----+
bool ChannelDelete(const long chart_ID=0, // chart's ID
                  const string name="Channel") // channel name
{
//--- reset the error value
    ResetLastError();
//--- delete the channel
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete the channel! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check the values of the channel's anchor points and set default values |
//| for empty ones |
//+-----+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
{
//--- if the second (right) point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, move it 300 points higher than the second
    if(!price1)
        price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the first point's one
    if(!time3)
        time3=time1;
//--- if the third point's price is not set, it is equal to the second point's one
    if(!price3)
        price3=price2;
}

```



```

    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the channel
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- create the equidistant channel
    if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price

```



```

        InpStyle, InpWidth, InpFill, InpBack, InpSelection, InpRayLeft, InpRayRight, InpHidden,
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
    int h_steps=bars/6;
//--- move the second anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d2<bars-1)
            d2+=1;
        //--- move the point
        if(!ChannelPointChange(0, InpName, 1, date[d2], price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
//--- 1 second of delay
    Sleep(1000);
//--- move the first anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d1>1)
            d1-=1;
        //--- move the point
        if(!ChannelPointChange(0, InpName, 0, date[d1], price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
//--- 1 second of delay
    Sleep(1000);
//--- loop counter

```



```
int v_steps=accuracy/10;
//--- move the third anchor point
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p3>1)
        p3-=1;
    //--- move the point
    if(!ChannelPointChange(0,InpName,2,date[d3],price[p3]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- delete the channel from the chart
ChannelDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_STDDEVCHANNEL

Standard Deviation Channel.



Note

For Standard Deviation Channel, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly). The mode of filling the channel with color can also be set.

[OBJPROP_DEVIATION](#) property is used to change the value of the channel deviation.

Example

The following script creates and moves Standard Deviation Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Standard Deviation Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="StdDevChannel";    // Channel name
input int         InpDate1=10;                // 1 st point's date, %
input int         InpDate2=40;                // 2 nd point's date, %
input double      InpDeviation=1.0;           // Deviation
input color       InpColor=clrRed;            // Channel color
```



```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int              InpWidth=2;                // Width of channel lines
input bool             InpFill=false;              // Filling the channel with color
input bool             InpBack=false;              // Background channel
input bool             InpSelection=true;          // Highlight to move
input bool             InpRayLeft=false;           // Channel's continuation to the left
input bool             InpRayRight=false;          // Channel's continuation to the right
input bool             InpHidden=true;             // Hidden in the object list
input long             InpZOrder=0;                // Priority for mouse click
//+-----+
//| Create standard deviation channel by the given coordinates |
//+-----+
bool StdDevChannelCreate(const long      chart_ID=0,      // chart's ID
                        const string    name="Channel",   // channel name
                        const int       sub_window=0,     // subwindow index
                        datetime         time1=0,          // first point time
                        datetime         time2=0,          // second point time
                        const double     deviation=1.0,    // deviation
                        const color      clr=clrRed,       // channel color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel
                        const int        width=1,         // width of channel
                        const bool       fill=false,      // filling the channel
                        const bool       back=false,      // in the background
                        const bool       selection=true,   // highlight to move
                        const bool       ray_left=false,   // channel's continuation to the left
                        const bool       ray_right=false,  // channel's continuation to the right
                        const bool       hidden=true,      // hidden in the object list
                        const long       z_order=0)        // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeChannelEmptyPoints(time1,time2);
    //--- reset the error value
    ResetLastError();
    //--- create a channel by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,time2,0))
    {
        Print(__FUNCTION__,
              ": failed to create standard deviation channel! Error code = ",GetLastError());
        return(false);
    }
    //--- set deviation value affecting the channel width
    ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
    //--- set channel color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- enable (true) or disable (false) the mode of filling the channel

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the channel's anchor point |
//+-----+
bool StdDevChannelPointChange(const long   chart_ID=0,    // chart's ID
                             const string name="Channel", // channel name
                             const int    point_index=0,  // anchor point index
                             datetime     time=0)         // anchor point time coordinate
{
//--- if point time is not set, move the point to the current bar
    if(!time)
        time=TimeCurrent();
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,0))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change the channel's deviation |
//+-----+
bool StdDevChannelDeviationChange(const long   chart_ID=0,    // chart's ID
                                  const string name="Channel", // channel name
                                  const double deviation=1.0)   // deviation

```



```

{
//--- reset the error value
    ResetLastError();
//--- change trend line's slope angle
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
    {
        Print(__FUNCTION__,
            ": failed to change channel deviation! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete the channel |
//+-----+
bool StdDevChannelDelete(const long   chart_ID=0,    // chart's ID
                        const string name="Channel") // channel name
{
//--- reset the error value
    ResetLastError();
//--- delete the channel
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check the values of the channel's anchor points and set default values |
//| for empty ones |
//+-----+
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
}

```



```

    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 ||
        InpDate2<0 || InpDate2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the channel
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
//--- create standard deviation channel
    if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpColor,InpStyle,
        InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder)
    {
        return;
    }
//--- redraw the chart and wait for 1 second

```



```

    ChartRedraw();
    Sleep(1000);
//--- now, move the channel horizontally to the right and expand it
//--- loop counter
    int h_steps=bars/2;
//--- move the channel
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following values
        if(d1<bars-1)
            d1+=1;
        if(d2<bars-1)
            d2+=1;
        //--- move the anchor points
        if(!StdDevChannelPointChange(0,InpName,0,date[d1]))
            return;
        if(!StdDevChannelPointChange(0,InpName,1,date[d2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
//--- 1 second of delay
    Sleep(1000);
//--- loop counter
    double v_steps=InpDeviation*2;
//--- expand the channel
    for(double i=InpDeviation;i<v_steps;i+=10.0/accuracy)
    {
        if(!StdDevChannelDeviationChange(0,InpName,i))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
//--- 1 second of delay
    Sleep(1000);
//--- delete the channel from the chart
    StdDevChannelDelete(0,InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---

```



```
}
```


OBJ_REGRESSION

Linear Regression Channel.



Note

For Linear Regression Channel, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly). The mode of filling the channel with color can also be set.

Example

The following script creates and moves Linear Regression Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Linear Regression Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Regression"; // Channel name
input int         InpDate1=10;          // 1 st point's date, %
input int         InpDate2=40;          // 2 nd point's date, %
input color       InpColor=clrRed;      // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of channel lines
input int         InpWidth=2;           // Width of channel lines
```



```

input bool      InpFill=false;           // Filling the channel with color
input bool      InpBack=false;           // Background channel
input bool      InpSelection=true;        // Highlight to move
input bool      InpRayLeft=false;         // Channel's continuation to the left
input bool      InpRayRight=false;        // Channel's continuation to the right
input bool      InpHidden=true;           // Hidden in the object list
input long      InpZOrder=0;              // Priority for mouse click
//+-----+
//| Create Linear Regression Channel by the given coordinates |
//+-----+
bool RegressionCreate(const long      chart_ID=0,           // chart's ID
                     const string    name="Regression",    // channel name
                     const int       sub_window=0,          // subwindow index
                     datetime         time1=0,              // first point time
                     datetime         time2=0,              // second point time
                     const color      clr=clrRed,           // channel color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel line
                     const int       width=1,              // width of channel line
                     const bool       fill=false,           // filling the channel
                     const bool       back=false,           // in the background
                     const bool       selection=true,        // highlight to move
                     const bool       ray_left=false,        // channel's continuation to the left
                     const bool       ray_right=false,       // channel's continuation to the right
                     const bool       hidden=true,           // hidden in the object list
                     const long       z_order=0)             // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeRegressionEmptyPoints(time1,time2);
    //--- reset the error value
    ResetLastError();
    //--- create a channel by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,time1,0,time2,0))
    {
        Print(__FUNCTION__,
              ": failed to create linear regression channel! Error code = ",GetLastError());
        return(false);
    }
    //--- set channel color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- enable (true) or disable (false) the mode of filling the channel
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of highlighting the channel for moving
    //--- when creating a graphical object using ObjectCreate function, the object cannot

```



```

//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the channel's anchor point |
//+-----+
bool RegressionPointChange(const long   chart_ID=0,    // chart's ID
                           const string name="Channel", // channel name
                           const int   point_index=0, // anchor point index
                           datetime    time=0)        // anchor point time coordinate
{
//--- if point time is not set, move the point to the current bar
    if(!time)
        time=TimeCurrent();
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,0))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete the channel |
//+-----+
bool RegressionDelete(const long   chart_ID=0,    // chart's ID
                      const string name="Channel") // channel name
{
//--- reset the error value
    ResetLastError();
//--- delete the channel
    if(!ObjectDelete(chart_ID,name))
    {

```



```

        Print(__FUNCTION__,
              ": failed to delete the channel! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check the values of the channel's anchor points and set default values |
//| for empty ones                                                         |
//+-----+
void ChangeRegressionEmptyPoints(datetime &time1, datetime &time2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(), Period(), time2, 10, temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
}
//+-----+
//| Script program start function                                         |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 ||
       InpDate2<0 || InpDate2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0, CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date, bars);
    ArrayResize(price, accuracy);

```



```

//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of prices
//--- find the highest and lowest values of the chart
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing the channel
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
//--- create linear regression channel
if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,InpWidth,
    InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the channel horizontally to the right
//--- loop counter
int h_steps=bars/2;
//--- move the channel
for(int i=0;i<h_steps;i++)
{
    //--- use the following values
    if(d1<bars-1)
        d1+=1;
    if(d2<bars-1)
        d2+=1;
    //--- move the anchor points
    if(!RegressionPointChange(0,InpName,0,date[d1]))
        return;
    if(!RegressionPointChange(0,InpName,1,date[d2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay

```



```
        Sleep(50);  
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the channel from the chart  
    RegressionDelete(0, InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_PITCHFORK

Andrews' Pitchfork.



Note

For Andrews' Pitchfork, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Andrews' Pitchfork on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Andrews' Pitchfork\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Pitchfork";    // Pitchfork name
input int         InpDate1=14;            // 1 st point's date, %
input int         InpPrice1=40;           // 1 st point's price, %
input int         InpDate2=18;            // 2 nd point's date, %
input int         InpPrice2=50;           // 2 nd point's price, %
input int         InpDate3=18;            // 3 rd point's date, %
input int         InpPrice3=30;           // 3 rd point's price, %
```



```

input color      InpColor=clrRed;      // Pitchfork color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Style of pitchfork lines
input int        InpWidth=1;          // Width of pitchfork lines
input bool       InpBack=false;       // Background pitchfork
input bool       InpSelection=true;    // Highlight to move
input bool       InpRayLeft=false;     // Pitchfork's continuation to the left
input bool       InpRayRight=false;    // Pitchfork's continuation to the right
input bool       InpHidden=true;      // Hidden in the object list
input long       InpZOrder=0;         // Priority for mouse click
//+-----+
//| Create Andrews' Pitchfork by the given coordinates |
//+-----+
bool PitchforkCreate(const long      chart_ID=0,      // chart's ID
                    const string    name="Pitchfork", // pitchfork name
                    const int       sub_window=0,    // subwindow index
                    datetime         time1=0,        // first point time
                    double           price1=0,       // first point price
                    datetime         time2=0,        // second point time
                    double           price2=0,       // second point price
                    datetime         time3=0,        // third point time
                    double           price3=0,       // third point price
                    const color      clr=clrRed,     // color of pitchfork lines
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of pitchfork lines
                    const int       width=1,        // width of pitchfork lines
                    const bool      back=false,     // in the background
                    const bool      selection=true,  // highlight to move
                    const bool      ray_left=false, // pitchfork's continuation to the left
                    const bool      ray_right=false, // pitchfork's continuation to the right
                    const bool      hidden=true,    // hidden in the object list
                    const long      z_order=0)      // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- create Andrews' Pitchfork by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create \"Andrews' Pitchfork\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the pitchfork for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the pitchfork's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the pitchfork's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Set number of Andrews' Pitchfork levels and their parameters |
//+-----+
bool PitchforkLevelsSet(int          levels,          // number of level lines
                        double        &values[],      // values of level lines
                        color          &colors[],      // color of level lines
                        ENUM_LINE_STYLE &styles[],     // style of level lines
                        int            &widths[],      // width of level lines
                        const long     chart_ID=0,     // chart's ID
                        const string   name="Pitchfork") // pitchfork name
{
//--- check array sizes
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
        levels!=ArraySize(widths) || levels!=ArraySize(values))
    {
        Print(__FUNCTION__,": array length does not correspond to the number of levels,
        return(false);
    }
//--- set the number of levels
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
    for(int i=0;i<levels;i++)
    {
        //--- level value
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- level color
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- level style
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- level width
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
    }
}

```



```

        //--- level description
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
    }
//--- successful execution
    return(true);
}
//+-----+
//| Move Andrews' Pitchfork anchor point |
//+-----+
bool PitchforkPointChange(const long   chart_ID=0,      // chart's ID
                          const string name="Pitchfork", // channel name
                          const int    point_index=0,    // anchor point index
                          datetime      time=0,          // anchor point time coordin
                          double        price=0)          // anchor point price coordin

{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete Andrews' Pitchfork |
//+-----+
bool PitchforkDelete(const long   chart_ID=0,      // chart's ID
                     const string name="Pitchfork") // channel name

{
//--- reset the error value
    ResetLastError();
//--- delete the channel
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Andrews' Pitchfork\"! Error code = ",GetLastError())
        return(false);
    }
//--- successful execution
    return(true);
}

```



```

    }
//+-----+
//| Check the values of Andrews' Pitchfork anchor points and set default |
//| values for empty ones |
//+-----+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                             double &price2,datetime &time3,double &price3)
{
//--- if the second (upper right) point's time is not set, it will be on the current k
    if(!time2)
        time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, move it 200 points below the second one
    if(!price1)
        price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second point's one
    if(!time3)
        time3=time2;
//--- if the third point's price is not set, move it 200 points lower than the first c
    if(!price3)
        price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size

```



```

    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing the coordinates of Andrews' Pitchfork anchor points
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- define points for drawing Andrews' Pitchfork
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
    //--- create the pitchfork
    if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price[p3],
        InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden))
    {
        return;
    }
    //--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //--- now, move the pitchfork's anchor points
    //--- loop counter
    int v_steps=accuracy/10;
    //--- move the first anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
    }

```



```

        if(!PitchforkPointChange(0, InpName, 0, date[d1], price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    int h_steps=bars/8;
    //--- move the third anchor point
    for(int i=0; i<h_steps; i++)
    {
        //--- use the following value
        if(d3<bars-1)
            d3+=1;
        //--- move the point
        if(!PitchforkPointChange(0, InpName, 2, date[d3], price[p3]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy/10;
    //--- move the second anchor point
    for(int i=0; i<v_steps; i++)
    {
        //--- use the following value
        if(p2>1)
            p2-=1;
        //--- move the point
        if(!PitchforkPointChange(0, InpName, 1, date[d2], price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }

```



```
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the pitchfork from the chart  
    PitchforkDelete(0, InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_GANNLIN

Gann Line.



Note

For Gann Line, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

Both Gann angle with a scale and coordinates of the second anchor point can be used to set the slope of the line.

Example

The following script creates and moves Gann Line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Gann Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="GannLine";           // Line name
input int         InpDate1=20;                  // 1 st point's date, %
input int         InpPrice1=75;                 // 1 st point's price, %
input int         InpDate2=80;                 // 2 nd point's date, %
input double      InpAngle=0.0;                // Gann Angle
```



```

input double      InpScale=1.0;           // Scale
input color       InpColor=clrRed;        // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int         InpWidth=2;             // Line width
input bool        InpBack=false;          // Background line
input bool        InpSelection=true;       // Highlight to move
input bool        InpRayLeft=false;        // Line's continuation to the left
input bool        InpRayRight=true;       // Line's continuation to the right
input bool        InpHidden=true;         // Hidden in the object list
input long        InpZOrder=0;            // Priority for mouse click

//+-----+
//| Create Gann Line by the coordinates, angle and scale |
//+-----+
bool GannLineCreate(const long      chart_ID=0,      // chart's ID
                    const string    name="GannLine", // line name
                    const int       sub_window=0,    // subwindow index
                    datetime         time1=0,         // first point time
                    double           price1=0,        // first point price
                    datetime         time2=0,         // second point time
                    const double     angle=1.0,       // Gann angle
                    const double     scale=1.0,       // scale
                    const color      clr=clrRed,      // line color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                    const int        width=1,         // line width
                    const bool       back=false,      // in the background
                    const bool       selection=true,   // highlight to move
                    const bool       ray_left=false,  // line's continuation to
                    const bool       ray_right=true,  // line's continuation to
                    const bool       hidden=true,     // hidden in the object list
                    const long       z_order=0)       // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeGannLineEmptyPoints(time1,price1,time2);
    //--- reset the error value
    ResetLastError();
    //--- create Gann Line by the given coordinates
    //--- correct coordinate of the second anchor point is redefined
    //--- automatically after Gann angle and/or the scale changes,
    if(!ObjectCreate(chart_ID,name,OBJ_GANNLIN,sub_window,time1,price1,time2,0))
    {
        Print(__FUNCTION__,
              ": failed to create \"Gann Line\"! Error code = ",GetLastError());
        return(false);
    }
    //--- change Gann angle
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
    //--- change the scale (number of pips per bar)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
    //--- set line color

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the lines for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move Gann Line anchor point |
//+-----+
bool GannLinePointChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannLine",  // line name
                        const int    point_index=0,    // anchor point index
                        datetime      time=0,          // anchor point time coordinate
                        double        price=0)          // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the line's anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```



```

    }
//+-----+
//| Change Gann angle |
//+-----+
bool GannLineAngleChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannLine",  // line name
                        const double angle=1.0)        // Gann angle
{
//--- reset the error value
    ResetLastError();
//--- change Gann angle
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
    {
        Print(__FUNCTION__,
              ": failed to change Gann angle! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Change Gann Line's scale |
//+-----+
bool GannLineScaleChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannLine",  // line name
                        const double scale=1.0)        // scale
{
//--- reset the error value
    ResetLastError();
//--- change the scale (number of pips per bar)
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
    {
        Print(__FUNCTION__,
              ": failed to change the scale! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| The function removes Gann Line from the chart |
//+-----+
bool GannLineDelete(const long   chart_ID=0,      // chart's ID
                   const string name="GannLine") // line name
{
//--- reset the error value
    ResetLastError();
//--- delete Gann line
    if(!ObjectDelete(chart_ID,name))

```



```

    {
        Print(__FUNCTION__,
              ": failed to delete \"Gann Line\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check the values of Gann Line anchor points and set default      |
//| values for empty ones                                           |
//+-----+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function                                    |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
    datetime date[];

```



```

    double   price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing Gann Line
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Line
    if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,InpScale,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the line's anchor point and change the angle
//--- loop counter
    int v_steps=accuracy/2;
//--- move the first anchor point vertically
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
        if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }

```



```
    }  
    //--- half a second of delay  
    Sleep(500);  
    //--- define the current value of Gann angle (changed  
    //--- after moving the first anchor point)  
    double curr_angle;  
    if(!ObjectGetDouble(0, InpName, OBJPROP_ANGLE, 0, curr_angle))  
        return;  
    //--- loop counter  
    v_steps=accuracy/8;  
    //--- change Gann angle  
    for(int i=0; i<v_steps; i++)  
    {  
        if(!GannLineAngleChange(0, InpName, curr_angle-0.05*i))  
            return;  
        //--- check if the script's operation has been forcefully disabled  
        if(IsStopped())  
            return;  
        //--- redraw the chart  
        ChartRedraw();  
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the line from the chart  
    GannLineDelete(0, InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_GANNFAN

Gann Fan.



Note

For Gann Fan, it is possible to specify trend type from [ENUM_GANN_DIRECTION](#) enumeration. By adjusting the scale value ([OBJPROP_SCALE](#)), it is possible to change slope angle of the fan lines.

Example

The following script creates and moves Gann Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Gann Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="GannFan";           // Fan name
input int         InpDate1=15;                 // 1 st point's date, %
input int         InpPrice1=25;                // 1 st point's price, %
input int         InpDate2=85;                // 2 nd point's date, %
input double      InpScale=2.0;                // Scale
input bool        InpDirection=false;         // Trend direction
input color       InpColor=clrRed;            // Fan color
```



```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of fan lines
input int              InpWidth=1;                // Width of fan lines
input bool             InpBack=false;             // Background fan
input bool             InpSelection=true;          // Highlight to move
input bool             InpHidden=true;            // Hidden in the object list
input long             InpZOrder=0;               // Priority for mouse click
//+-----+
//| Create Gann Fan                                     |
//+-----+
bool GannFanCreate(const long      chart_ID=0,      // chart's ID
                  const string    name="GannFan",  // fan name
                  const int       sub_window=0,    // subwindow index
                  datetime        time1=0,         // first point time
                  double          price1=0,        // first point price
                  datetime        time2=0,         // second point time
                  const double    scale=1.0,       // scale
                  const bool      direction=true,   // trend direction
                  const color     clr=clrRed,      // fan color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // style of fan lines
                  const int       width=1,         // width of fan lines
                  const bool      back=false,      // in the background
                  const bool      selection=true,   // highlight to move
                  const bool      hidden=true,     // hidden in the object list
                  const long      z_order=0)       // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeGannFanEmptyPoints(time1,price1,time2);
    //--- reset the error value
    ResetLastError();
    //--- create Gann Fan by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time2,0))
    {
        Print(__FUNCTION__,
              ": failed to create \"Gann Fan\"! Error code = ",GetLastError());
        return(false);
    }
    //--- change the scale (number of pips per bar)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
    //--- change Gann Fan's trend direction (true - descending, false - ascending)
    ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
    //--- set fan color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set display style of the fan lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the fan lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of highlighting the fan for moving

```



```

//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move Gann Fan anchor point |
//+-----+
bool GannFanPointChange(const long   chart_ID=0,    // chart's ID
                        const string name="GannFan", // fan name
                        const int    point_index=0, // anchor point index
                        datetime      time=0,       // anchor point time coordinate
                        double        price=0)       // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the fan's anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Gann Fan's scale |
//+-----+
bool GannFanScaleChange(const long   chart_ID=0,    // chart's ID
                        const string name="GannFan", // fan name
                        const double scale=1.0)      // scale
{
//--- reset the error value
    ResetLastError();
//--- change the scale (number of pips per bar)
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))

```



```

    {
        Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Gann Fan's trend direction |
//+-----+
bool GannFanDirectionChange(const long   chart_ID=0,    // chart's ID
                           const string name="GannFan", // fan name
                           const bool   direction=true) // trend direction
{
//--- reset the error value
    ResetLastError();
//--- change Gann Fan's trend direction
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
    {
        Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| The function removes Gann Fan from the chart |
//+-----+
bool GannFanDelete(const long   chart_ID=0,    // chart's ID
                  const string name="GannFan") // fan name
{
//--- reset the error value
    ResetLastError();
//--- delete Gann Fan
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete \"Gann Fan\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
// | Check the values of Gann Fan anchor points and set default |
//| values for empty ones |
//+-----+

```



```

void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
    //--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
    //--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
    //--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

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

void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing the coordinates of fan's anchor points
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices

```



```

//--- find the highest and lowest values of the chart
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing Gann Fan
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Fan
if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the fan's anchor point
//--- loop counter
int v_steps=accuracy/2;
//--- move the first anchor point vertically
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p1<accuracy-1)
        p1+=1;
    //--- move the point
    if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- change fan's trend direction to descending one
GannFanDirectionChange(0,InpName,true);
//--- redraw the chart
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//--- delete the fan from the chart
GannFanDelete(0,InpName);
ChartRedraw();

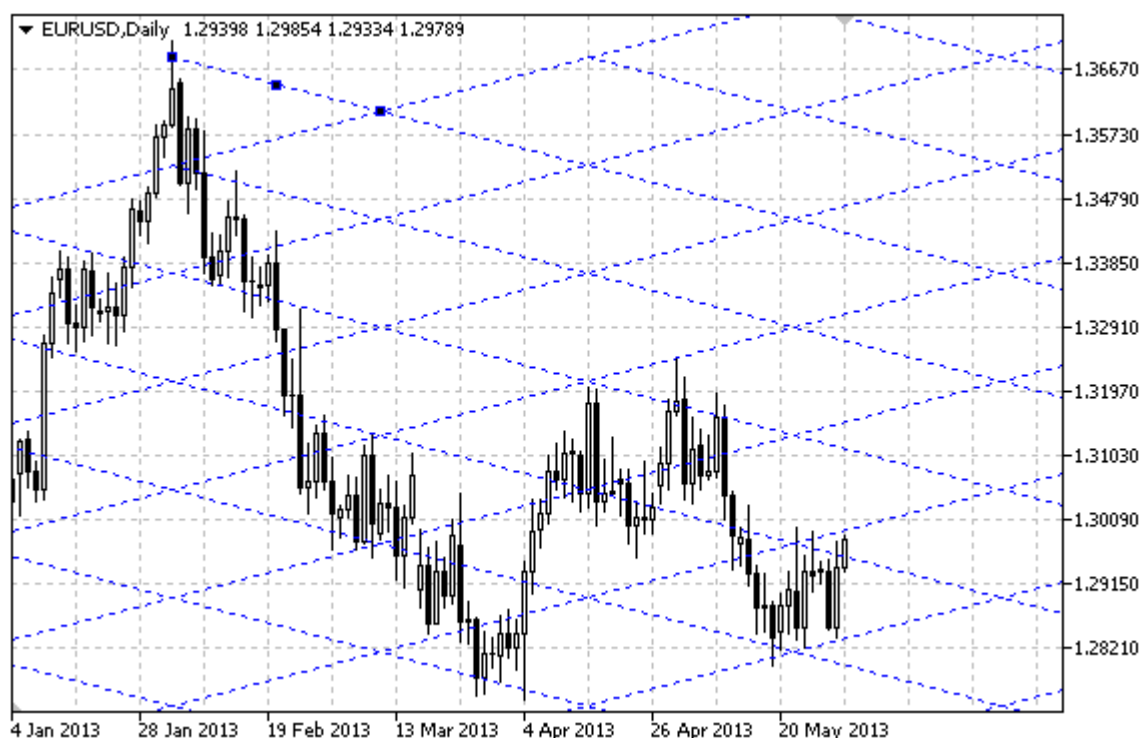
```



```
//--- 1 second of delay  
    Sleep(1000);  
//---  
}
```


OBJ_GANNGRID

Gann Grid.



Note

For Gann Grid, it is possible to specify trend type from [ENUM_GANN_DIRECTION](#). By adjusting the scale value ([OBJPROP_SCALE](#)), it is possible to change slope angle of the grid lines.

Example

The following script creates and moves Gann Grid on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Gann Grid\" graphical object."
#property description "Anchor point coordinates of the grid are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="GannGrid";           // Grid name
input int         InpDate1=15;                  // 1 st point's date, %
input int         InpPrice1=25;                 // 1 st point's price, %
input int         InpDate2=35;                 // 2 nd point's date, %
input double      InpScale=3.0;                 // Scale
input bool        InpDirection=false;          // Trend direction
input color       InpColor=clrRed;             // Grid color
```



```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of grid lines
input int              InpWidth=1;                // Width of fan lines
input bool             InpBack=false;             // Background grid
input bool             InpSelection=true;         // Highlight to move
input bool             InpHidden=true;           // Hidden in the object list
input long             InpZOrder=0;              // Priority for mouse click
//+-----+
//| Create Gann Grid                                     |
//+-----+
bool GannGridCreate(const long      chart_ID=0,      // chart's ID
                   const string    name="GannGrid", // grid name
                   const int       sub_window=0,    // subwindow index
                   datetime         time1=0,        // first point time
                   double           price1=0,        // first point price
                   datetime         time2=0,        // second point time
                   const double     scale=1.0,      // scale
                   const bool       direction=true, // trend direction
                   const color      clr=clrRed,     // grid color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of grid lines
                   const int        width=1,        // width of grid lines
                   const bool        back=false,    // in the background
                   const bool        selection=true, // highlight to move
                   const bool        hidden=true,   // hidden in the object list
                   const long        z_order=0)     // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeGannGridEmptyPoints(time1,price1,time2);
    //--- reset the error value
    ResetLastError();
    //--- create Gann Grid by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,time1,price1,time2,0))
    {
        Print(__FUNCTION__,
              ": failed to create \"Gann Grid\"! Error code = ",GetLastError());
        return(false);
    }
    //--- change the scale (number of pips per bar)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
    //--- change Gann Fan's trend direction (true - descending, false - ascending)
    ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
    //--- set grid color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set display style of the grid lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the grid lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of highlighting the grid for moving

```



```

//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move Gann Grid anchor point |
//+-----+
bool GannGridPointChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannGrid",  // grid name
                        const int    point_index=0,    // anchor point index
                        datetime      time=0,          // anchor point time coordinate
                        double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the grid's anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Gann Grid's scale |
//+-----+
bool GannGridScaleChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannGrid",  // grids
                        const double scale=1.0)        // scale
{
//--- reset the error value
    ResetLastError();
//--- change the scale (number of pips per bar)
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))

```



```

    {
        Print(__FUNCTION__,
            ": failed to change the scale! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Gann Grid's trend direction |
//+-----+
bool GannGridDirectionChange(const long   chart_ID=0,      // chart's ID
                             const string name="GannGrid", // grid name
                             const bool   direction=true)   // trend direction
{
//--- reset the error value
    ResetLastError();
//--- change Gann Grid's trend direction
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
    {
        Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| The function removes Gann Fan from the chart |
//+-----+
bool GannGridDelete(const long   chart_ID=0,      // chart's ID
                    const string name="GannGrid") // grid name
{
//--- reset the error value
    ResetLastError();
//--- delete Gann Grid
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete \"Gann Grid\"! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check the values of Gann Grid anchor points and set default |
//| values for empty ones |
//+-----+

```



```

void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
//--- if the second point's time is not set, it will be on the current bar
if(!time2)
    time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
if(!time1)
{
    //--- array for receiving the open time of the last 10 bars
    datetime temp[10];
    CopyTime(Symbol(),Period(),time2,10,temp);
    //--- set the first point 9 bars left from the second one
    time1=temp[0];
}
//--- if the first point's price is not set, it will have Bid value
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
    InpDate2<0 || InpDate2>100)
{
    Print("Error! Incorrect values of input parameters!");
    return;
}
//--- number of visible bars in the chart window
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing grid anchor points' coordinates
datetime date[];
double price[];
//--- memory allocation
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of prices

```



```

//--- find the highest and lowest values of the chart
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing Gann Grid
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Grid
if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the grid's anchor points
//--- loop counter
int v_steps=accuracy/4;
//--- move the first anchor point vertically
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p1<accuracy-1)
        p1+=1;
    if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
int h_steps=bars/4;
//--- move the second anchor point horizontally
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d2<bars-1)
        d2+=1;
    if(!GannGridPointChange(0,InpName,1,date[d2],0))
        return;
}

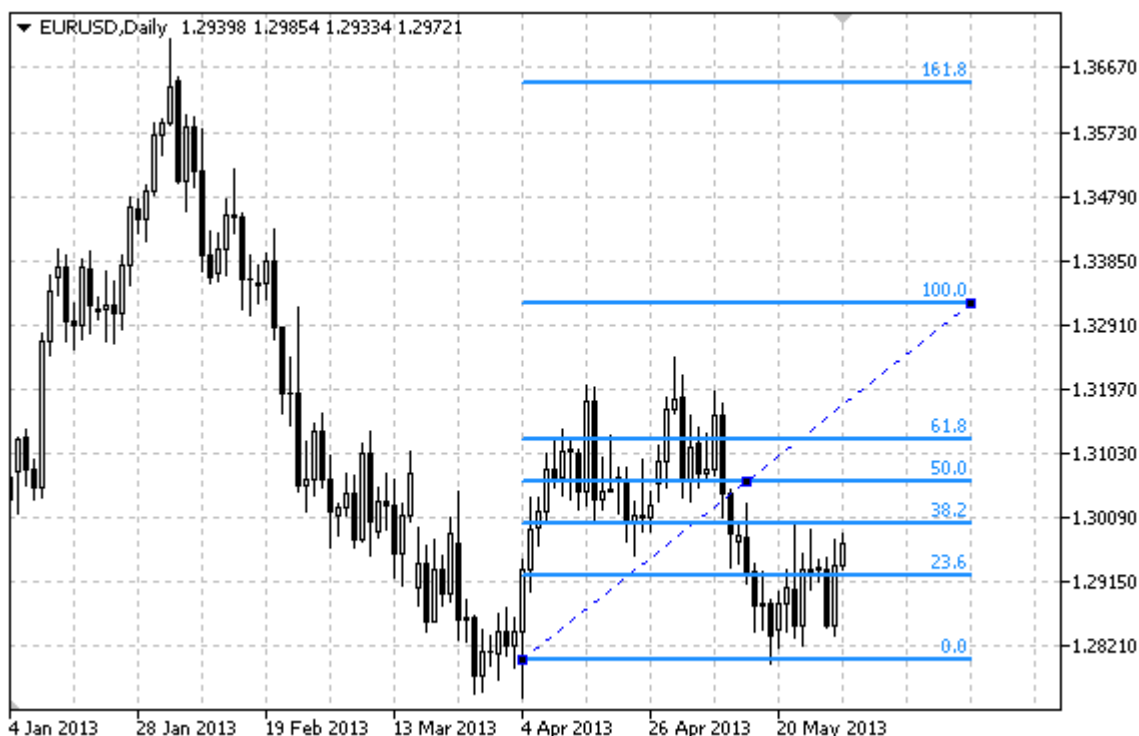
```



```
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- change grid's trend direction to descending one
GannGridDirectionChange(0, InpName, true);
//--- redraw the chart
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//--- delete the grid from the chart
GannGridDelete(0, InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_FIBO

Fibonacci Retracement.



Note

For Fibonacci Retracement, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Retracement on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Retracement\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboLevels";           // Object name
input int         InpDate1=10;                    // 1 st point's date, %
input int         InpPrice1=65;                    // 1 st point's price, %
input int         InpDate2=90;                    // 2 nd point's date, %
input int         InpPrice2=85;                    // 2 nd point's price, %
input color       InpColor=clrRed;                 // Object color
```



```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int              InpWidth=2;                // Line width
input bool             InpBack=false;             // Background object
input bool             InpSelection=true;         // Highlight to move
input bool             InpRayLeft=false;          // Object's continuation to the left
input bool             InpRayRight=false;         // Object's continuation to the right
input bool             InpHidden=true;            // Hidden in the object list
input long             InpZOrder=0;               // Priority for mouse click
//+-----+
//| Create Fibonacci Retracement by the given coordinates |
//+-----+
bool FiboLevelsCreate(const long      chart_ID=0,      // chart's ID
                     const string    name="FiboLevels", // object name
                     const int       sub_window=0,    // subwindow index
                     datetime         time1=0,         // first point time
                     double           price1=0,        // first point price
                     datetime         time2=0,         // second point time
                     double           price2=0,        // second point price
                     const color      clr=clrRed,      // object color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                     const int       width=1,         // object line width
                     const bool      back=false,      // in the background
                     const bool      selection=true,  // highlight to move
                     const bool      ray_left=false,  // object's continuation to the left
                     const bool      ray_right=false, // object's continuation to the right
                     const bool      hidden=true,     // hidden in the object list
                     const long      z_order=0)        // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);
    //--- reset the error value
    ResetLastError();
    //--- Create Fibonacci Retracement by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Retracement\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of highlighting the channel for moving
    //--- when creating a graphical object using ObjectCreate function, the object cannot

```



```

//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the object's display
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the object's display
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}

//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboLevelsSet(int          levels,          // number of level lines
                  double        &values[],      // values of level lines
                  color          &colors[],      // color of level lines
                  ENUM_LINE_STYLE &styles[],     // style of level lines
                  int            &widths[],     // width of level lines
                  const long     chart_ID=0,     // chart's ID
                  const string   name="FiboLevels") // object name
{
//--- check array sizes
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
    levels!=ArraySize(widths) || levels!=ArraySize(values))
{
    Print(__FUNCTION__,": array length does not correspond to the number of levels,
    return(false);
}

//--- set the number of levels
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
for(int i=0;i<levels;i++)
{
    //--- level value
    ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
    //--- level color
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
    //--- level style
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
    //--- level width
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
    //--- level description
    ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
}

```



```

//--- successful execution
    return(true);
}

//+-----+
//| Move Fibonacci Retracement anchor point |
//+-----+
bool FiboLevelsPointChange(const long   chart_ID=0,      // chart's ID
                           const string name="FiboLevels", // object name
                           const int   point_index=0,    // anchor point index
                           datetime    time=0,          // anchor point time coordi
                           double      price=0)          // anchor point price coordi

{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete Fibonacci Retracement |
//+-----+
bool FiboLevelsDelete(const long   chart_ID=0,      // chart's ID
                      const string name="FiboLevels") // object name

{
//--- reset the error value
    ResetLastError();
//--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Fibonacci Retracement\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check the values of Fibonacci Retracement anchor points and set |

```



```

//| default values for empty ones |
//+-----+
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, move it 200 points below the second one
    if(!price1)
        price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Retracement anchor points
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();

```



```

    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- define points for drawing Fibonacci Retracement
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    //--- create an object
    if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder)
    {
        return;
    }
    //--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //--- now, move the anchor points
    //--- loop counter
    int v_steps=accuracy*2/5;
    //--- move the first anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
        if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy*4/5;

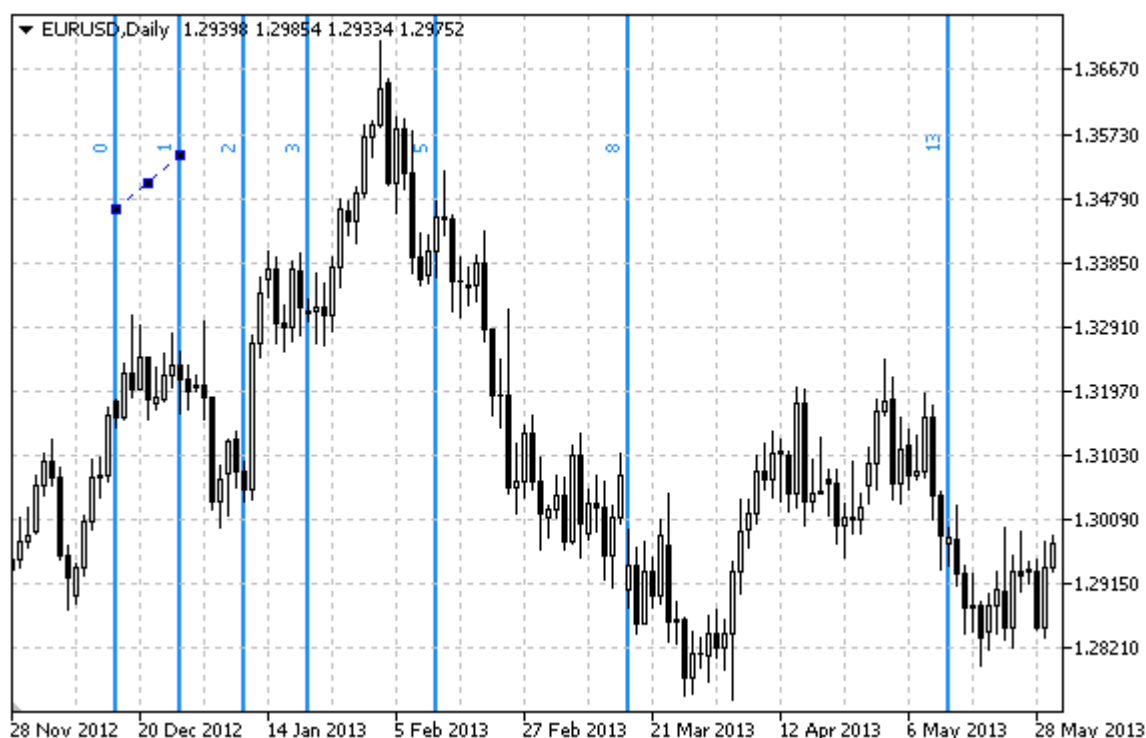
```



```
//--- move the second anchor point
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p2>1)
        p2-=1;
    //--- move the point
    if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- delete the object from the chart
FiboLevelsDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_FIBOTIMES

Fibonacci Time Zones.



Note

For "Fibonacci Time Zones", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Time Zones on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Time Zones\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboTimes";           // Object name
input int         InpDate1=10;                   // 1 st point's date, %
input int         InpPrice1=45;                  // 1 st point's price, %
input int         InpDate2=20;                   // 2 nd point's date, %
input int         InpPrice2=55;                  // 2 nd point's price, %
input color       InpColor=clrRed;               // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int         InpWidth=2;                    // Line width
```



```

input bool      InpBack=false;           // Background object
input bool      InpSelection=true;       // Highlight to move
input bool      InpHidden=true;         // Hidden in the object list
input long      InpZOrder=0;            // Priority for mouse click
//+-----+
//| Create Fibonacci Time Zones by the given coordinates |
//+-----+
bool FiboTimesCreate(const long      chart_ID=0,      // chart's ID
                    const string     name="FiboTimes", // object name
                    const int        sub_window=0,   // subwindow index
                    datetime          time1=0,        // first point time
                    double            price1=0,       // first point price
                    datetime          time2=0,        // second point time
                    double            price2=0,       // second point price
                    const color       clr=clrRed,     // object color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                    const int         width=1,       // object line width
                    const bool        back=false,    // in the background
                    const bool        selection=true, // highlight to move
                    const bool        hidden=true,   // hidden in the object
                    const long        z_order=0)     // priority for mouse click
{
//--- set anchor points' coordinates if they are not set
    ChangeFiboTimesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
    ResetLastError();
//--- create Fibonacci Time Zones by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Time Zones\"! Error code = ",GetLastError()
              return(false);
    }
//--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

```



```

//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboTimesLevelsSet(int          levels,          // number of level lines
                        double       &values[],       // values of level lines
                        color        &colors[],       // color of level lines
                        ENUM_LINE_STYLE &styles[],     // style of level lines
                        int          &widths[],       // width of level lines
                        const long    chart_ID=0,      // chart's ID
                        const string  name="FiboTimes") // object name
{
//--- check array sizes
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
        levels!=ArraySize(widths) || levels!=ArraySize(widths))
    {
        Print(__FUNCTION__,": array length does not correspond to the number of levels,
        return(false);
    }
//--- set the number of levels
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
    for(int i=0;i<levels;i++)
    {
        //--- level value
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- level color
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- level style
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- level width
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
        //--- level description
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(values[i],1));
    }
//--- successful execution
    return(true);
}

//+-----+
//| Move Fibonacci Time Zones anchor point |
//+-----+
bool FiboTimesPointChange(const long    chart_ID=0,      // chart's ID
                          const string  name="FiboTimes", // object name
                          const int     point_index=0,    // anchor point index
                          datetime      time=0,          // anchor point time coordinat

```



```

        double price=0) // anchor point price coordin

{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
        ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Delete Fibonacci Time Zones |
//+-----+
bool FiboTimesDelete(const long chart_ID=0, // chart's ID
                    const string name="FiboTimes") // object name
{
//--- reset the error value
ResetLastError();
//--- delete the object
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
        ": failed to delete \"Fibonacci Time Zones\"! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Check the values of Fibonacci Time Zones and |
//| set default values for empty ones |
//+-----+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
if(!time1)
    time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
if(!price1)

```



```

    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 2 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 3 bars
        datetime temp[3];
        CopyTime(Symbol(),Period(),time1,3,temp);
        //--- set the first point 2 bars left from the second one
        time2=temp[0];
    }
//--- if the second point's price is not set, it is equal to the first point's one
    if(!price2)
        price2=price1;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }

//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Time Zones anchor points
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array

```



```

double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Time Zones
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
if(!FiboTimesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the anchor points
//--- loop counter
int h_steps=bars*2/5;
//--- move the second anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d2<bars-1)
        d2+=1;
    //--- move the point
    if(!FiboTimesPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
h_steps=bars*3/5;
//--- move the first anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d1<bars-1)
        d1+=1;
    //--- move the point
    if(!FiboTimesPointChange(0,InpName,0,date[d1],price[p1]))

```



```
        return;

        ///--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;

        ///--- redraw the chart
        ChartRedraw();

        /// 0.05 seconds of delay
        Sleep(50);
    }

    ///--- 1 second of delay
    Sleep(1000);

    ///--- delete the object from the chart
    FiboTimesDelete(0, InpName);

    ChartRedraw();

    ///--- 1 second of delay
    Sleep(1000);

    ///---
}
```


OBJ_FIBOFAN

Fibonacci Fan.



Note

For "Fibonacci Fan", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboFan";           // Fan name
input int         InpDate1=10;                 // 1 st point's date, %
input int         InpPrice1=25;                // 1 st point's price, %
input int         InpDate2=30;                // 2 nd point's date, %
input int         InpPrice2=50;               // 2 nd point's price, %
input color       InpColor=clrRed;            // Fan line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int         InpWidth=2;                 // Line width
```



```

input bool      InpBack=false;           // Background object
input bool      InpSelection=true;       // Highlight to move
input bool      InpHidden=true;         // Hidden in the object list
input long      InpZOrder=0;            // Priority for mouse click
//+-----+
//| Create Fibonacci Fan by the given coordinates |
//+-----+
bool FiboFanCreate(const long      chart_ID=0,           // chart's ID
                  const string    name="FiboFan",       // fan name
                  const int       sub_window=0,         // subwindow index
                  datetime        time1=0,              // first point time
                  double          price1=0,              // first point price
                  datetime        time2=0,              // second point time
                  double          price2=0,              // second point price
                  const color      clr=clrRed,          // fan line color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // fan line style
                  const int       width=1,              // fan line width
                  const bool      back=false,           // in the background
                  const bool      selection=true,       // highlight to move
                  const bool      hidden=true,          // hidden in the object list
                  const long      z_order=0)            // priority for mouse click
{
//--- set anchor points' coordinates if they are not set
    ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
    ResetLastError();
//--- create Fibonacci Fan by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Fan\"! Error code = ",GetLastError());
        return(false);
    }
//--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the fan for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

```



```

//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}
//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboFanLevelsSet(int          levels,          // number of level lines
                     double        &values[],       // values of level lines
                     color         &colors[],       // color of level lines
                     ENUM_LINE_STYLE &styles[],      // style of level lines
                     int           &widths[],       // width of level lines
                     const long     chart_ID=0,      // chart's ID
                     const string   name="FiboFan") // fan name
{
//--- check array sizes
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
    levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
    Print(__FUNCTION__,": array length does not correspond to the number of levels,
    return(false);
}
//--- set the number of levels
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
for(int i=0;i<levels;i++)
{
    //--- level value
    ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
    //--- level color
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
    //--- level style
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
    //--- level width
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
    //--- level description
    ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],2));
}
//--- successful execution
return(true);
}
//+-----+
//| Move Fibonacci Fan anchor point |
//+-----+
bool FiboFanPointChange(const long     chart_ID=0,      // chart's ID
                       const string   name="FiboFan", // fan name
                       const int      point_index=0,   // anchor point index
                       datetime        time=0,         // anchor point time coordinate

```



```

                                double      price=0)      // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete Fibonacci Fan |
//+-----+
bool FiboFanDelete(const long   chart_ID=0,      // chart's ID
                  const string name="FiboFan") // fan name
{
//--- reset the error value
    ResetLastError();
//--- delete the fan
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Fibonacci Fan\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check the values of Fibonacci Fan anchor points and set |
//| default values for empty ones |
//+-----+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
    if(!price2)

```



```

        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, move it 200 points below the second one
    if(!price1)
        price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }

    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing the coordinates of Fibonacci Fan anchor points
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }

    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array

```



```

double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Fan
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
if(!FiboFanCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the fan's anchor points
//--- loop counter
int v_steps=accuracy/2;
//--- move the first anchor point
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p1<accuracy-1)
        p1+=1;
    //--- move the point
    if(!FiboFanPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
int h_steps=bars/4;
//--- move the second anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d2<bars-1)
        d2+=1;
    //--- move the point
    if(!FiboFanPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled

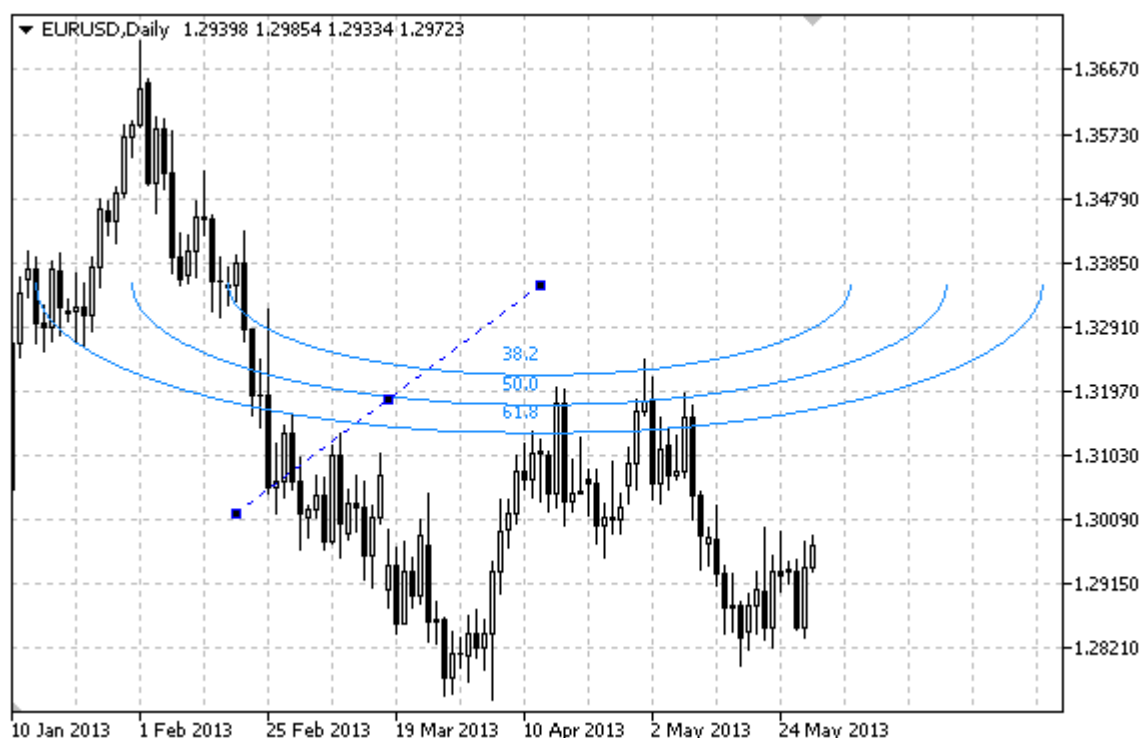
```



```
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the object from the chart
    FiboFanDelete(0, InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}
```


OBJ_FIBOARC

Fibonacci Arcs.



Note

For "Fibonacci Arcs", it is possible to specify the display mode of the entire ellipse. Curvature radius can be specified by changing the scale and coordinates of the anchor points.

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Arcs on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Arcs\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboArc";           // Object name
input int         InpDate1=25;                 // 1 st point's date, %
input int         InpPrice1=25;                // 1 st point's price, %
input int         InpDate2=35;                // 2 nd point's date, %
input int         InpPrice2=55;               // 2 nd point's price, %
input double      InpScale=3.0;                // Scale
```



```

input bool      InpFullEllipse=true;      // Shape of the arcs
input color     InpColor=clrRed;          // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int       InpWidth=2;               // Line width
input bool      InpBack=false;            // Background object
input bool      InpSelection=true;        // Highlight to move
input bool      InpHidden=true;          // Hidden in the object list
input long      InpZOrder=0;              // Priority for mouse click
//+-----+
//| Create Fibonacci Arcs by the given coordinates |
//+-----+
bool FiboArcCreate(const long      chart_ID=0,      // chart's ID
                  const string    name="FiboArc",  // object name
                  const int       sub_window=0,    // subwindow index
                  datetime        time1=0,         // first point time
                  double          price1=0,        // first point price
                  datetime        time2=0,         // second point time
                  double          price2=0,        // second point price
                  const double    scale=1.0,       // scale
                  const bool      full_ellipse=false, // shape of the arcs
                  const color     clr=clrRed,      // line color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                  const int       width=1,         // line width
                  const bool      back=false,      // in the background
                  const bool      selection=true,   // highlight to move
                  const bool      hidden=true,     // hidden in the object list
                  const long      z_order=0)       // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
    //--- reset the error value
    ResetLastError();
    //--- create Fibonacci Arcs by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Arcs\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set the scale
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
    //--- set display of the arcs as a full ellipse (true) or a half of it (false)
    ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
    //--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set line width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

```



```

//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the arcs for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboArcLevelsSet(int          levels,          // number of level lines
                     double        &values[],       // values of level lines
                     color          &colors[],      // color of level lines
                     ENUM_LINE_STYLE &styles[],     // style of level lines
                     int            &widths[],      // width of level lines
                     const long     chart_ID=0,     // chart's ID
                     const string   name="FiboArc") // object name
{
//--- check array sizes
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
        levels!=ArraySize(widths) || levels!=ArraySize(values))
    {
        Print(__FUNCTION__,": array length does not correspond to the number of levels,
        return(false);
    }
//--- set the number of levels
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
    for(int i=0;i<levels;i++)
    {
        //--- level value
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- level color
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- level style
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- level width
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
        //--- level description
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
    }

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Move Fibonacci Arcs anchor point |
//+-----+
bool FiboArcPointChange(const long   chart_ID=0,    // chart's ID
                        const string name="FiboArc", // object name
                        const int    point_index=0, // anchor point index
                        datetime      time=0,       // anchor point time coordinate
                        double         price=0)      // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete Fibonacci Arcs |
//+-----+
bool FiboArcDelete(const long   chart_ID=0,    // chart's ID
                   const string name="FiboArc") // object name
{
//--- reset the error value
    ResetLastError();
//--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Fibonacci Arcs\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check the values of Fibonacci Arcs anchor points and set default |

```



```

//| values for empty ones |
//+-----+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
{
//--- if the second point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
//--- if the first point's price is not set, move it 300 points below the second one
    if(!price1)
        price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Arcs anchor points
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();

```



```

    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- define points for drawing Fibonacci Arcs
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    //--- create an object
    if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpScale,
        InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrd
    {
        return;
    }
    //--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //--- now, move the anchor points
    //--- loop counter
    int v_steps=accuracy/5;
    //--- move the first anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1<accuracy-1)
            p1+=1;
        //--- move the point
        if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    int h_steps=bars/5;

```



```
//--- move the second anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d2<bars-1)
        d2+=1;
    //--- move the point
    if(!FiboArcPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- delete the object from the chart
FiboArcDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_FIBOCHANNEL

Fibonacci Channel.



Note

For Fibonacci Channel, it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboChannel";      // Channel name
input int         InpDate1=20;                // 1 st point's date, %
input int         InpPrice1=10;               // 1 st point's price, %
input int         InpDate2=60;               // 2 nd point's date, %
input int         InpPrice2=30;              // 2 nd point's price, %
input int         InpDate3=20;              // 3 rd point's date, %
```



```

input int      InpPrice3=25;           // 3 rd point's price, %
input color    InpColor=clrRed;        // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int      InpWidth=2;             // Width of channel lines
input bool     InpBack=false;          // Background channel
input bool     InpSelection=true;       // Highlight to move
input bool     InpRayLeft=false;       // Channel's continuation to the left
input bool     InpRayRight=false;      // Channel's continuation to the right
input bool     InpHidden=true;         // Hidden in the object list
input long     InpZOrder=0;            // Priority for mouse click

//+-----+
//| Create Fibonacci Channel by the given coordinates |
//+-----+

bool FiboChannelCreate(const long      chart_ID=0,           // chart's ID
                      const string    name="FiboChannel",   // channel name
                      const int       sub_window=0,         // subwindow index
                      datetime         time1=0,             // first point time
                      double           price1=0,            // first point price
                      datetime         time2=0,             // second point time
                      double           price2=0,            // second point price
                      datetime         time3=0,             // third point time
                      double           price3=0,            // third point price
                      const color      clr=clrRed,          // channel color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel lines
                      const int        width=1,            // width of channel lines
                      const bool       back=false,         // in the background
                      const bool       selection=true,      // highlight to move
                      const bool       ray_left=false,     // channel's continuation to the left
                      const bool       ray_right=false,    // channel's continuation to the right
                      const bool       hidden=true,         // hidden in the object list
                      const long       z_order=0)           // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- create a channel by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Channel\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set channel color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the channel lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

```



```

//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's display
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboChannelLevelsSet(int          levels,          // number of level lines
                        double         &values[],       // values of level lines
                        color           &colors[],       // color of level lines
                        ENUM_LINE_STYLE &styles[],       // style of level lines
                        int             &widths[],       // width of level lines
                        const long      chart_ID=0,      // chart's ID
                        const string    name="FiboChannel") // object name
{
//--- check array sizes
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
        levels!=ArraySize(widths) || levels!=ArraySize(values))
    {
        Print(__FUNCTION__,": array length does not correspond to the number of levels,
        return(false);
    }
//--- set the number of levels
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
    for(int i=0;i<levels;i++)
    {
        //--- level value
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- level color
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- level style
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- level width

```



```

        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
        //--- level description
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Move Fibonacci Channel anchor point |
//+-----+
bool FiboChannelPointChange(const long   chart_ID=0,           // chart's ID
                           const string name="FiboChannel",    // channel name
                           const int    point_index=0,         // anchor point index
                           datetime      time=0,               // anchor point time coordinate
                           double        price=0)               // anchor point price coordinate
{
    //--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- reset the error value
    ResetLastError();
    //--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Delete the channel |
//+-----+
bool FiboChannelDelete(const long   chart_ID=0,           // chart's ID
                      const string name="FiboChannel")    // channel name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the channel
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Fibonacci Channel\"! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution

```



```

    return(true);
}

//+-----+
//| Check the values of Fibonacci Channel anchor points and set |
//| default values for empty ones                               |
//+-----+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                   double &price2,datetime &time3,double &price3)
{
    //--- if the second (right) point's time is not set, it will be on the current bar
    if(!time2)
        time2=TimeCurrent();
    //--- if the second point's price is not set, it will have Bid value
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the first (left) point's time is not set, it is located 9 bars left from the
    if(!time1)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
    //--- if the first point's price is not set, move it 300 points higher than the second
    if(!price1)
        price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    //--- if the third point's time is not set, it coincides with the first point's one
    if(!time3)
        time3=time1;
    //--- if the third point's price is not set, it is equal to the second point's one
    if(!price3)
        price3=price2;
}

//+-----+
//| Script program start function                               |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }

    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

```



```

//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the channel
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Channel
    if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
        InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
    int h_steps=bars/10;
//--- move the first anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d1>1)
            d1-=1;

```



```

    //--- move the point
    if(!FiboChannelPointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
int v_steps=accuracy/10;
//--- move the second anchor point
for(int i=0; i<v_steps; i++)
{
    //--- use the following value
    if(p2>1)
        p2-=1;
    //--- move the point
    if(!FiboChannelPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
v_steps=accuracy/15;
//--- move the third anchor point
for(int i=0; i<v_steps; i++)
{
    //--- use the following value
    if(p3<accuracy-1)
        p3+=1;
    //--- move the point
    if(!FiboChannelPointChange(0, InpName, 2, date[d3], price[p3]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}

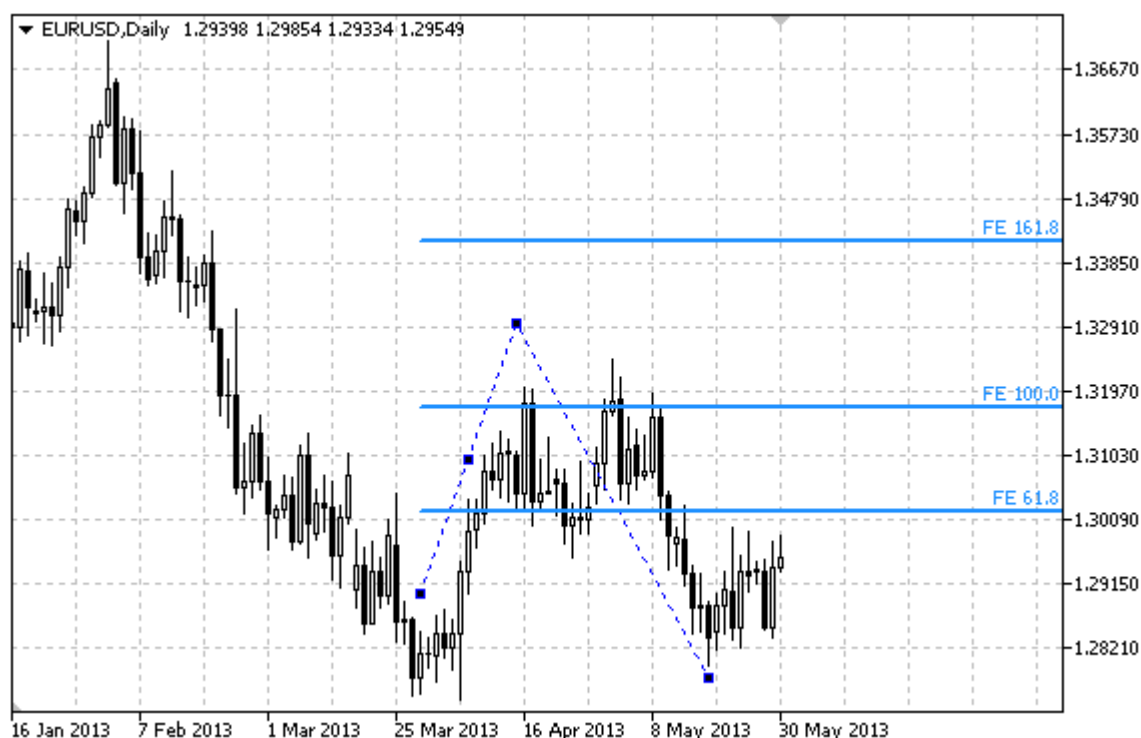
```



```
//--- 1 second of delay
    Sleep(1000);
//--- delete the channel from the chart
    FiboChannelDelete(0, InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---
}
```


OBJ_EXPANSION

Fibonacci Expansion.



Note

For "Fibonacci Expansion", it is possible to specify the mode of continuation of its display to the right and/or left ([OBJPROP_RAY_RIGHT](#) and [OBJPROP_RAY_LEFT](#) properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Expansion on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Expansion\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="FiboExpansion";    // Object name
input int         InpDate1=10;                // 1 st point's date, %
input int         InpPrice1=55;               // 1 st point's price, %
input int         InpDate2=30;               // 2 nd point's date, %
input int         InpPrice2=10;              // 2 nd point's price, %
input int         InpDate3=80;              // 3 rd point's date, %
```



```

input int      InpPrice3=75;           // 3 rd point's price, %
input color    InpColor=clrRed;        // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of lines
input int      InpWidth=2;             // Width of the lines
input bool     InpBack=false;          // Background object
input bool     InpSelection=true;       // Highlight to move
input bool     InpRayLeft=false;       // Object's continuation to the left
input bool     InpRayRight=false;      // Object's continuation to the right
input bool     InpHidden=true;         // Hidden in the object list
input long     InpZOrder=0;            // Priority for mouse click

//+-----+
//| Create Fibonacci Extension by the given coordinates |
//+-----+
bool FiboExpansionCreate(const long      chart_ID=0,           // chart's ID
                        const string     name="FiboExpansion", // channel name
                        const int        sub_window=0,        // subwindow index
                        datetime         time1=0,             // first point time
                        double           price1=0,            // first point price
                        datetime         time2=0,             // second point time
                        double           price2=0,            // second point price
                        datetime         time3=0,             // third point time
                        double           price3=0,            // third point price
                        const color       clr=clrRed,         // object color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // style of the line
                        const int        width=1,            // width of the line
                        const bool       back=false,         // in the background
                        const bool       selection=true,      // highlight to move
                        const bool       ray_left=false,      // object's continuation to the left
                        const bool       ray_right=false,     // object's continuation to the right
                        const bool       hidden=true,         // hidden in the object list
                        const long       z_order=0)           // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- Create Fibonacci Extension by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create \"Fibonacci Extension\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set the object's color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

```



```

//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the object's visual
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the object's visual
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Set number of levels and their parameters |
//+-----+
bool FiboExpansionLevelsSet(int          levels,          // number of level
                           double        &values[],       // values of level
                           color         &colors[],       // color of level
                           ENUM_LINE_STYLE &styles[],     // style of level
                           int           &widths[],       // width of level
                           const long    chart_ID=0,      // chart's ID
                           const string   name="FiboExpansion") // object name
{
//--- check array sizes
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
        levels!=ArraySize(widths) || levels!=ArraySize(values))
    {
        Print(__FUNCTION__,": array length does not correspond to the number of levels,
        return(false);
    }
//--- set the number of levels
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
    for(int i=0;i<levels;i++)
    {
        //--- level value
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- level color
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- level style
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- level width

```



```

        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
        //--- level description
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToString(100*value
    }
//--- successful execution
    return(true);
}
//+-----+
//| Move Fibonacci Expansion anchor point |
//+-----+
bool FiboExpansionPointChange(const long   chart_ID=0,           // chart's ID
                             const string name="FiboExpansion", // object name
                             const int    point_index=0,        // anchor point index
                             datetime      time=0,              // anchor point time
                             double        price=0)              // anchor point price
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete Fibonacci Expansion |
//+-----+
bool FiboExpansionDelete(const long   chart_ID=0,           // chart's ID
                        const string name="FiboExpansion") // object name
{
//--- reset the error value
    ResetLastError();
//--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Fibonacci Expansion\"! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution

```



```

    return(true);
}

//+-----+
//| Check the values of Fibonacci Expansion anchor points and set |
//| default values for empty ones                                |
//+-----+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                     double &price2,datetime &time3,double &price3)
{
    //--- if the third (right) point's time is not set, it will be on the current bar
    if(!time3)
        time3=TimeCurrent();
    //--- if the third point's price is not set, it will have Bid value
    if(!price3)
        price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the first (left) point's time is not set, it is located 9 bars left from the
    //--- array for receiving the open time of the last 10 bars
    datetime temp[];
    ArrayResize(temp,10);
    if(!time1)
    {
        CopyTime(Symbol(),Period(),time3,10,temp);
        //--- set the first point 9 bars left from the second one
        time1=temp[0];
    }
    //--- if the first point's price is not set, it is equal to the third point's one
    if(!price1)
        price1=price3;
    //--- if the second point's time is not set, it is located 7 bars left from the third
    if(!time2)
        time2=temp[2];
    //--- if the second point's price is not set, move it 250 points lower than the first
    if(!price2)
        price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}

//+-----+
//| Script program start function                                |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
}

//--- number of visible bars in the chart window

```



```

    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Expansion
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Expansion
    if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],
        InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden)
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor points
//--- loop counter
    int v_steps=accuracy/10;
//--- move the first anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)

```



```

        p1-=1;
        //--- move the point
        if(!FiboExpansionPointChange(0, InpName, 0, date[d1], price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy/2;
    //--- move the third anchor point
    for(int i=0; i<v_steps; i++)
    {
        //--- use the following value
        if(p3>1)
            p3-=1;
        //--- move the point
        if(!FiboExpansionPointChange(0, InpName, 2, date[d3], price[p3]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy*4/5;
    //--- move the second anchor point
    for(int i=0; i<v_steps; i++)
    {
        //--- use the following value
        if(p2<accuracy-1)
            p2+=1;
        //--- move the point
        if(!FiboExpansionPointChange(0, InpName, 1, date[d2], price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay

```



```
Sleep(1000);  
//--- delete the object from the chart  
FiboExpansionDelete(0, InpName);  
ChartRedraw();  
//--- 1 second of delay  
Sleep(1000);  
//---  
}
```


OBJ_ELLIOTWAVE5

Elliott Motive Wave.



Note

For "Elliott Motive Wave", it is possible to enable/disable the mode of connecting points by lines ([OBJPROP_DRAWLINES](#) property), as well as set the level of wave positioning (from [ENUM_ELLIOT_WAVE_DEGREE](#) enumeration).

Example

The following script creates and moves Elliott motive wave on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Elliott Motive Wave\"."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ElliottWave5";    // Object name
input int         InpDate1=10;               // 1 st point's date, %
input int         InpPrice1=90;              // 1 st point's price, %
input int         InpDate2=20;               // 2 nd point's date, %
input int         InpPrice2=40;              // 2 nd point's price, %
input int         InpDate3=30;              // 3 rd point's date, %
```



```

input int                InpPrice3=60;                // 3 rd point's price, %
input int                InpDate4=40;                // 4 th point's date, %
input int                InpPrice4=10;                // 4 th point's price, %
input int                InpDate5=60;                // 5 th point's date, %
input int                InpPrice5=40;                // 5 th point's price, %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Level
input bool              InpDrawLines=true;           // Displaying the lines
input color             InpColor=clrRed;             // Color of the lines
input ENUM_LINE_STYLE   InpStyle=STYLE_DASH;         // Style of the lines
input int               InpWidth=2;                 // Width of the lines
input bool              InpBack=false;              // Background object
input bool              InpSelection=true;           // Highlight to move
input bool              InpHidden=true;             // Hidden in the object list
input long              InpZOrder=0;                // Priority for mouse click
//+-----+
//| Create "Elliott Motive Wave" by the given coordinates |
//+-----+
bool ElliotWave5Create(const long      chart_ID=0,           // chart
                      const string    name="ElliotWave5",    // wave
                      const int       sub_window=0,          // subwindow
                      datetime        time1=0,              // first
                      double           price1=0,             // first
                      datetime         time2=0,              // second
                      double           price2=0,             // second
                      datetime         time3=0,              // third
                      double           price3=0,             // third
                      datetime         time4=0,              // fourth
                      double           price4=0,             // fourth
                      datetime         time5=0,              // fifth
                      double           price5=0,             // fifth
                      const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // degree
                      const bool      draw_lines=true,       // display
                      const color      clr=clrRed,           // object
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // style
                      const int        width=1,              // width
                      const bool       back=false,           // in the
                      const bool       selection=true,       // highlight
                      const bool       hidden=true,          // hidden
                      const long       z_order=0)            // priority
{
//--- set anchor points' coordinates if they are not set
    ChangeElliotWave5EmptyPoints(time1,price1,time2,price2,time3,price3,time4,price4,time5,price5);
//--- reset the error value
    ResetLastError();
//--- Create "Elliott Motive Wave" by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE5,sub_window,time1,price1,time2,price2,price3,time4,price4,time5,price5))
    {
        Print(__FUNCTION__,

```



```

        ": failed to create \"Elliott Motive Wave\"! Error code = ", GetLastError());
    return(false);
}

//--- set degree (wave size)
ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- enable (true) or disable (false) the mode of displaying the lines
ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- set the object's color
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the line style
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}

//+-----+
//| Move anchor point of Elliott Motive Wave |
//+-----+
bool ElliotWave5PointChange(const long   chart_ID=0,          // chart's ID
                           const string name="ElliotWave5",    // object name
                           const int    point_index=0,         // anchor point index
                           datetime      time=0,               // anchor point time coordinate
                           double        price=0)               // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
        ": failed to move the anchor point! Error code = ",GetLastError());
}
}

```



```

        return(false);
    }
    //--- successful execution
    return(true);
}
//+-----+
//| Delete Elliott Motive Wave |
//+-----+
bool ElliotWave5Delete(const long   chart_ID=0,          // chart's ID
                      const string name="ElliotWave5") // object name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Elliott Motive Wave\"! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
//+-----+
//| Check the values of Elliott Motive Wave's anchor points and |
//| set default values for empty ones |
//+-----+
void ChangeElliotWave5EmptyPoints(datetime &time1,double &price1,
                                   datetime &time2,double &price2,
                                   datetime &time3,double &price3,
                                   datetime &time4,double &price4,
                                   datetime &time5,double &price5)
{
    //--- array for receiving the open time of the last 10 bars
    datetime temp[];
    ArrayResize(temp,10);
    //--- receive data
    CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
    //--- receive the value of one point on the current chart
    double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    //--- if the first point's time is not set, it will be 9 bars left from the last bar
    if(!time1)
        time1=temp[0];
    //--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the second point's time is not set, it will be 7 bars left from the last bar
    if(!time2)
        time2=temp[2];
}

```



```

//--- if the second point's price is not set, move it 300 points lower than the first
    if(!price2)
        price2=price1-300*point;
//--- if the third point's time is not set, it will be 5 bars left from the last bar
    if(!time3)
        time3=temp[4];
//--- if the third point's price is not set, move it 250 points lower than the first c
    if(!price3)
        price3=price1-250*point;
//--- if the fourth point's time is not set, it will be 3 bars left from the last bar
    if(!time4)
        time4=temp[6];
//--- if the fourth point's price is not set, move it 550 points lower than the first
    if(!price4)
        price4=price1-550*point;
//--- if the fifth point's time is not set, it will be on the last bar
    if(!time5)
        time5=temp[9];
//--- if the fifth point's price is not set, move it 450 points lower than the first c
    if(!price5)
        price5=price1-450*point;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100 ||
        InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
        InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates

```



```

ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of prices
//--- find the highest and lowest values of the chart
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- define points for drawing Elliott Motive Wave
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int d4=InpDate4*(bars-1)/100;
int d5=InpDate5*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
int p4=InpPrice4*(accuracy-1)/100;
int p5=InpPrice5*(accuracy-1)/100;
//--- Create Elliott Motive Wave
if(!ElliotWave5Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price[p3],date[d4],price[p4],date[d5],price[p5],InpDegree,InpDrawLines,InpColor,InpStyle,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- now, move the anchor points
//--- loop counter
int v_steps=accuracy/5;
//--- move the fifth anchor point
for(int i=0;i<v_steps;i++)
{
    //--- use the following value
    if(p5<accuracy-1)
        p5+=1;
    //--- move the point
    if(!ElliotWave5PointChange(0,InpName,4,date[d5],price[p5]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())

```



```

        return;

        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy/5;
    //--- move the second and third anchor points
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following values
        if(p2<accuracy-1)
            p2+=1;
        if(p3>1)
            p3-=1;
        //--- shift the points
        if(!ElliotWave5PointChange(0,InpName,1,date[d2],price[p2]))
            return;
        if(!ElliotWave5PointChange(0,InpName,2,date[d3],price[p3]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy*4/5;
    //--- move the first and fourth anchor points
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following values
        if(p1>1)
            p1-=1;
        if(p4<accuracy-1)
            p4+=1;
        //--- shift the points
        if(!ElliotWave5PointChange(0,InpName,0,date[d1],price[p1]))
            return;
        if(!ElliotWave5PointChange(0,InpName,3,date[d4],price[p4]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }

```



```
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the object from the chart  
    ElliotWave5Delete(0, InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_ELLIOTWAVE3

Elliott Correction Wave.



Note

For "Elliott Correction Wave", it is possible to enable/disable the mode of connecting points by lines ([OBJPROP_DRAWLINES](#) property), as well as set the level of wave positioning (from [ENUM_ELLIOT_WAVE_DEGREE](#) enumeration).

Example

The following script creates and moves Elliott correction wave on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Elliott Correction Wave\" graphical object."
#property description "Anchor point coordinates are set in percentage of the chart's v
#property description "size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ElliottWave3";    // Object name
input int         InpDate1=10;               // 1 st point's date, %
input int         InpPrice1=90;              // 1 st point's price, %
input int         InpDate2=30;              // 2 nd point's date, %
input int         InpPrice2=10;             // 2 nd point's price, %
input int         InpDate3=50;              // 3 rd point's date, %
```



```

input int                InpPrice3=40;           // 3 rd point's price, %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Level
input bool               InpDrawLines=true;      // Displaying the lines
input color              InpColor=clrRed;        // Color of the lines
input ENUM_LINE_STYLE    InpStyle=STYLE_DASH;    // Style of the lines
input int                InpWidth=2;            // Width of the lines
input bool               InpBack=false;          // Background object
input bool               InpSelection=true;      // Highlight to move
input bool               InpHidden=true;         // Hidden in the object list
input long               InpZOrder=0;           // Priority for mouse click

//+-----+
//| Create "Elliott Correction Wave" by the given coordinates |
//+-----+

bool ElliotWave3Create(const long      chart_ID=0,           // chart
                      const string    name="ElliotWave3",    // wave
                      const int       sub_window=0,         // subwindow
                      datetime         time1=0,             // first point
                      double           price1=0,            // first price
                      datetime         time2=0,             // second point
                      double           price2=0,            // second price
                      datetime         time3=0,             // third point
                      double           price3=0,            // third price
                      const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // degree
                      const bool       draw_lines=true,     // displaying lines
                      const color      clr=clrRed,          // object color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // style
                      const int        width=1,             // width
                      const bool       back=false,          // in the background
                      const bool       selection=true,       // highlight
                      const bool       hidden=true,          // hidden
                      const long       z_order=0)           // priority
{
    //--- set anchor points' coordinates if they are not set
    ChangeElliotWave3EmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- Create "Elliott Correction Wave" by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE3,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create \"Elliott Correction Wave\"! Error code = ",GetLastError());
        return(false);
    }
    //--- set degree (wave size)
    ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
    //--- enable (true) or disable (false) the mode of displaying the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
    //--- set the object's color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

```



```

//--- set the line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move anchor point of Elliott Correction Wave |
//+-----+
bool ElliotWave3PointChange(const long   chart_ID=0,          // chart's ID
                           const string name="ElliotWave3",    // object name
                           const int    point_index=0,         // anchor point index
                           datetime      time=0,               // anchor point time coordinate
                           double        price=0)               // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete Elliott Correction Wave |
//+-----+
bool ElliotWave3Delete(const long   chart_ID=0,          // chart's ID

```



```

        const string name="ElliotWave3") // object name

    {
//--- reset the error value
        ResetLastError();
//--- delete the object
        if(!ObjectDelete(chart_ID,name))
        {
            Print(__FUNCTION__,
                ": failed to delete \"Elliott Correction Wave\"! Error code = ",GetLastError());
            return(false);
        }
//--- successful execution
        return(true);
    }
//+-----+
//| Check the values of Elliott Correction Wave's anchor points |
//| and set default values for empty ones |
//+-----+
void ChangeElliotWave3EmptyPoints(datetime &time1,double &price1,
                                   datetime &time2,double &price2,
                                   datetime &time3,double &price3)

{
//--- array for receiving the open time of the last 10 bars
    datetime temp[];
    ArrayResize(temp,10);
//--- receive data
    CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- receive the value of one point on the current chart
    double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the first point's time is not set, it will be 9 bars left from the last bar
    if(!time1)
        time1=temp[0];
//--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it will be 5 bars left from the last bar
    if(!time2)
        time2=temp[4];
//--- if the second point's price is not set, move it 300 points lower than the first
    if(!price2)
        price2=price1-300*point;
//--- if the third point's time is not set, it will be 1 bar left from the last bar
    if(!time3)
        time3=temp[8];
//--- if the third point's price is not set, move it 200 points lower than the first
    if(!price3)
        price3=price1-200*point;
}
//+-----+

```



```

//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing Elliott Correction Wave
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- Create Elliott Correction Wave
    if(!ElliotWave3Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
        InpDegree,InpDrawLines,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHider
    {

```



```

        return;
    }
    //--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //--- now, move the anchor points
    //--- loop counter
    int v_steps=accuracy/5;
    //--- move the third anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p3<accuracy-1)
            p3+=1;
        //--- move the point
        if(!ElliotWave3PointChange(0,InpName,2,date[d3],price[p3]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    v_steps=accuracy*4/5;
    //--- move the first and second anchor points
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following values
        if(p1>1)
            p1-=1;
        if(p2<accuracy-1)
            p2+=1;
        //--- shift the points
        if(!ElliotWave3PointChange(0,InpName,0,date[d1],price[p1]))
            return;
        if(!ElliotWave3PointChange(0,InpName,1,date[d2],price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the object from the chart

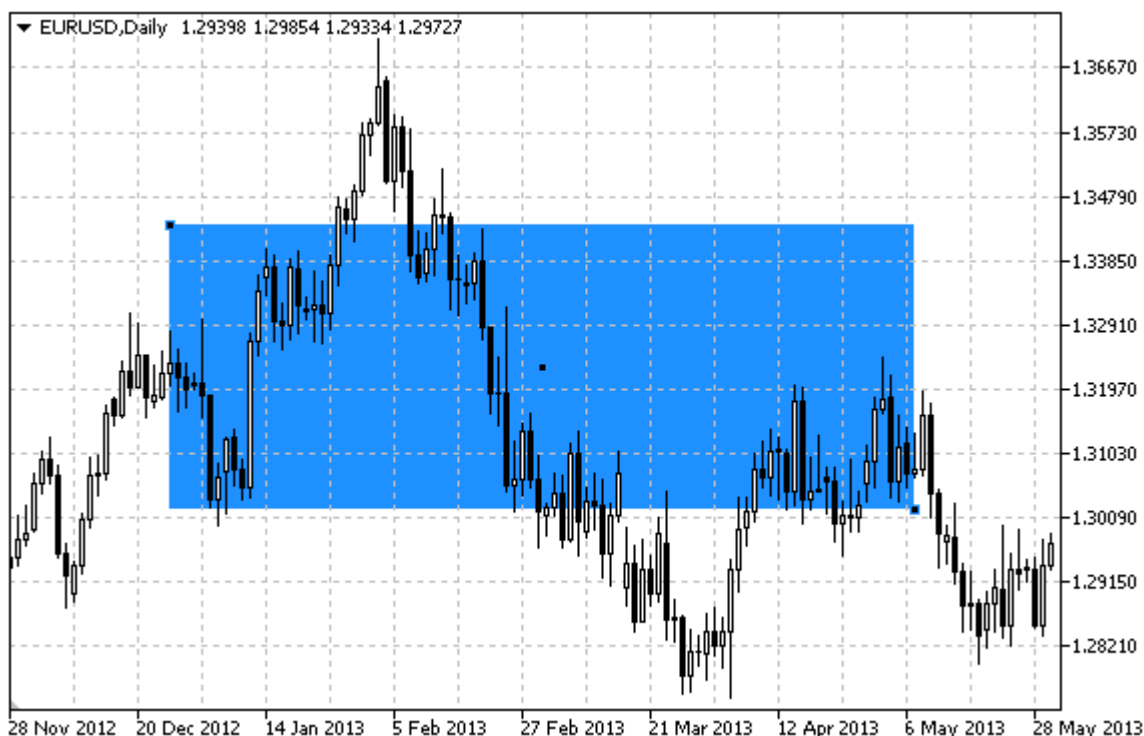
```



```
ElliotWave3Delete(0, InpName);  
ChartRedraw();  
//--- 1 second of delay  
Sleep(1000);  
//---  
}
```


OBJ_RECTANGLE

Rectangle.



Note

For rectangle, the mode of filling with color can be set using [OBJPROP_FILL](#) property.

Example

The following script creates and moves the rectangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates rectangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Rectangle"; // Rectangle name
input int         InpDate1=40;         // 1 st point's date, %
input int         InpPrice1=40;        // 1 st point's price, %
input int         InpDate2=60;        // 2 nd point's date, %
input int         InpPrice2=60;        // 2 nd point's price, %
input color       InpColor=clrRed;     // Rectangle color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of rectangle lines
input int         InpWidth=2;          // Width of rectangle lines
```



```

input bool      InpFill=true;          // Filling the rectangle with color
input bool      InpBack=false;         // Background rectangle
input bool      InpSelection=true;     // Highlight to move
input bool      InpHidden=true;       // Hidden in the object list
input long      InpZOrder=0;          // Priority for mouse click
//+-----+
//| Create rectangle by the given coordinates |
//+-----+
bool RectangleCreate(const long      chart_ID=0,          // chart's ID
                    const string     name="Rectangle",    // rectangle name
                    const int        sub_window=0,        // subwindow index
                    datetime         time1=0,             // first point time
                    double            price1=0,           // first point price
                    datetime         time2=0,             // second point time
                    double            price2=0,           // second point price
                    const color      clr=clrRed,          // rectangle color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of rectangle lines
                    const int        width=1,            // width of rectangle lines
                    const bool       fill=false,         // filling rectangle with color
                    const bool       back=false,         // in the background
                    const bool       selection=true,     // highlight to move
                    const bool       hidden=true,        // hidden in the object list
                    const long       z_order=0)          // priority for mouse click
{
//--- set anchor points' coordinates if they are not set
    ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
    ResetLastError();
//--- create a rectangle by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": failed to create a rectangle! Error code = ",GetLastError());
        return(false);
    }
//--- set rectangle color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the style of rectangle lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the rectangle lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the rectangle
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the rectangle for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- be highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the rectangle anchor point |
//+-----+
bool RectanglePointChange(const long   chart_ID=0,      // chart's ID
                          const string name="Rectangle", // rectangle name
                          const int    point_index=0,   // anchor point index
                          datetime      time=0,          // anchor point time coordin
                          double        price=0)          // anchor point price coordin
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete the rectangle |
//+-----+
bool RectangleDelete(const long   chart_ID=0,      // chart's ID
                     const string name="Rectangle") // rectangle name
{
//--- reset the error value
    ResetLastError();
//--- delete rectangle
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete rectangle! Error code = ",GetLastError());
        return(false);
    }
}

```



```

    }
    //--- successful execution
    return(true);
}
//+-----+
//| Check the values of rectangle's anchor points and set default |
//| values for empty ones                                         |
//+-----+
void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
{
    //--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
    //--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
    //--- if the second point's price is not set, move it 300 points lower than the first
    if(!price2)
        price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}
//+-----+
//| Script program start function                                  |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing rectangle anchor points' coordinates
    datetime date[];

```



```

    double   price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the rectangle
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
//--- create a rectangle
    if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
        InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the rectangle's anchor points
//--- loop counter
    int h_steps=bars/2;
//--- move the anchor points
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following values
        if(d1<bars-1)
            d1+=1;
        if(d2>1)
            d2-=1;
        //--- shift the points
        if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
            return;
        if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
            return;
    }

```



```

    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
int v_steps=accuracy/2;
//--- move the anchor points
for(int i=0;i<v_steps;i++)
{
    //--- use the following values
    if(p1<accuracy-1)
        p1+=1;
    if(p2>1)
        p2-=1;
    //--- shift the points
    if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
        return;
    if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- delete the rectangle from the chart
RectangleDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}

```


OBJ_TRIANGLE

Triangle.



Note

For triangle, the mode of filling with color can be set using [OBJPROP_FILL](#) property.

Example

The following script creates and moves the triangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates triangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Triangle";           // Triangle name
input int         InpDate1=25;                  // 1 st point's date, %
input int         InpPrice1=50;                 // 1 st point's price, %
input int         InpDate2=70;                 // 2 nd point's date, %
input int         InpPrice2=70;                 // 2 nd point's price, %
input int         InpDate3=65;                 // 3 rd point's date, %
input int         InpPrice3=20;                 // 3 rd point's price, %
input color       InpColor=clrRed;              // Triangle color
```



```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of triangle lines
input int              InpWidth=2;                // Width of triangle lines
input bool             InpFill=false;              // Filling triangle with color
input bool             InpBack=false;              // Background triangle
input bool             InpSelection=true;          // Highlight to move
input bool             InpHidden=true;             // Hidden in the object list
input long             InpZOrder=0;                // Priority for mouse click
//+-----+
//| Create triangle by the given coordinates |
//+-----+
bool TriangleCreate(const long      chart_ID=0,      // chart's ID
                   const string    name="Triangle", // triangle name
                   const int       sub_window=0,    // subwindow index
                   datetime         time1=0,        // first point time
                   double           price1=0,       // first point price
                   datetime         time2=0,        // second point time
                   double           price2=0,       // second point price
                   datetime         time3=0,        // third point time
                   double           price3=0,       // third point price
                   const color      clr=clrRed,     // triangle color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of triangle line
                   const int       width=1,        // width of triangle line
                   const bool      fill=false,     // filling triangle with
                   const bool      back=false,     // in the background
                   const bool      selection=true,  // highlight to move
                   const bool      hidden=true,    // hidden in the object list
                   const long      z_order=0)      // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeTriangleEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- create triangle by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create a triangle! Error code = ",GetLastError());
        return(false);
    }
    //--- set triangle color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of triangle lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of triangle lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- enable (true) or disable (false) the mode of filling the triangle
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
}

```



```

//--- enable (true) or disable (false) the mode of highlighting the triangle for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the triangle anchor point |
//+-----+
bool TrianglePointChange(const long   chart_ID=0,      // chart's ID
                        const string name="Triangle",  // triangle name
                        const int    point_index=0,    // anchor point index
                        datetime      time=0,          // anchor point time coordinate
                        double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete the triangle |
//+-----+
bool TriangleDelete(const long   chart_ID=0,      // chart's ID
                   const string name="Triangle") // triangle name
{
//--- reset the error value
    ResetLastError();
//--- delete the triangle
    if(!ObjectDelete(chart_ID,name))

```



```

    {
        Print(__FUNCTION__,
            ": failed to delete the ellipse! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check the values of triangle's anchor points and set default |
//| values for empty ones |
//+-----+
void ChangeTriangleEmptyPoints(datetime &time1, double &price1,
                                datetime &time2, double &price2,
                                datetime &time3, double &price3)
{
//--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(), SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(), Period(), time1, 10, temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
//--- if the second point's price is not set, move it 300 points lower than the first
    if(!price2)
        price2=price1-300*SymbolInfoDouble(Symbol(), SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second point's date
    if(!time3)
        time3=time2;
//--- if the third point's price is not set, it is equal to the first point's one
    if(!price3)
        price3=price1;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||

```



```

    InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing triangle anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the triangle
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- create a triangle
    if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price[p3],
        InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the triangle anchor points
//--- loop counter

```



```

    int v_steps=accuracy*3/10;
//--- move the first anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p1>1)
            p1-=1;
        //--- move the point
        if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
//--- 1 second of delay
    Sleep(1000);
//--- loop counter
    int h_steps=bars*9/20-1;
//--- move the second anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d2>1)
            d2-=1;
        //--- move the point
        if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
//--- 1 second of delay
    Sleep(1000);
//--- loop counter
    v_steps=accuracy/4;
//--- move the third anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p3<accuracy-1)
            p3+=1;
        //--- move the point
        if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))

```



```
        return;

        ///--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;

        ///--- redraw the chart
        ChartRedraw();
    }

    ///--- 1 second of delay
    Sleep(1000);

    ///--- delete triangle from the chart
    TriangleDelete(0, InpName);
    ChartRedraw();

    ///--- 1 second of delay
    Sleep(1000);

    ///---
}
```


OBJ_ELLIPSE

Ellipse.



Note

For ellipse, the mode of filling with color can be set using [OBJPROP_FILL](#) property.

Example

The following script creates and moves the ellipse on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates ellipse on the chart."
#property description "Anchor point coordinates are set"
#property description "in percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Ellipse";           // Ellipse name
input int         InpDate1=30;                  // 1 st point's date, %
input int         InpPrice1=20;                 // 1 st point's price, %
input int         InpDate2=70;                 // 2 nd point's date, %
input int         InpPrice2=80;                 // 2 nd point's price, %
input int         InpDate3=50;                 // 3 rd point's date, %
input int         InpPrice3=60;                 // 3 rd point's price, %
input color       InpColor=clrRed;              // Ellipse color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of ellipse lines
```



```

input int      InpWidth=2;           // Width of ellipse lines
input bool     InpFill=false;        // Filling ellipse with color
input bool     InpBack=false;        // Background ellipse
input bool     InpSelection=true;    // Highlight to move
input bool     InpHidden=true;       // Hidden in the object list
input long     InpZOrder=0;          // Priority for mouse click
//+-----+
//| Create an ellipse by the given coordinates |
//+-----+
bool EllipseCreate(const long      chart_ID=0,      // chart's ID
                   const string   name="Ellipse",  // ellipse name
                   const int       sub_window=0,    // subwindow index
                   datetime        time1=0,         // first point time
                   double          price1=0,        // first point price
                   datetime        time2=0,         // second point time
                   double          price2=0,        // second point price
                   datetime        time3=0,         // third point time
                   double          price3=0,        // third point price
                   const color     clr=clrRed,      // ellipse color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of ellipse lines
                   const int       width=1,        // width of ellipse lines
                   const bool      fill=false,     // filling ellipse with color
                   const bool      back=false,     // in the background
                   const bool      selection=true,  // highlight to move
                   const bool      hidden=true,    // hidden in the object list
                   const long      z_order=0)      // priority for mouse click
{
    //--- set anchor points' coordinates if they are not set
    ChangeEllipseEmptyPoints(time1,price1,time2,price2,time3,price3);
    //--- reset the error value
    ResetLastError();
    //--- create an ellipse by the given coordinates
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": failed to create an ellipse! Error code = ",GetLastError());
        return(false);
    }
    //--- set an ellipse color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set style of ellipse lines
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set width of ellipse lines
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- enable (true) or disable (false) the mode of filling the ellipse
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of highlighting the ellipse for moving

```



```

//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the ellipse anchor point |
//+-----+
bool EllipsePointChange(const long   chart_ID=0,    // chart's ID
                        const string name="Ellipse", // ellipse name
                        const int    point_index=0,  // anchor point index
                        datetime      time=0,        // anchor point time coordinate
                        double         price=0)       // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete ellipse |
//+-----+
bool EllipseDelete(const long   chart_ID=0,    // chart's ID
                   const string name="Ellipse") // ellipse name
{
//--- reset the error value
    ResetLastError();
//--- delete an ellipse
    if(!ObjectDelete(chart_ID,name))
    {

```



```

        Print(__FUNCTION__,
              ": failed to delete an ellipse! Error code = ", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Check the values of ellipse anchor points and set default values |
//| for empty ones |
//+-----+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2,
                              datetime &time3,double &price3)
{
    //--- if the first point's time is not set, it will be on the current bar
    if(!time1)
        time1=TimeCurrent();
    //--- if the first point's price is not set, it will have Bid value
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- if the second point's time is not set, it is located 9 bars left from the second
    if(!time2)
    {
        //--- array for receiving the open time of the last 10 bars
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
        //--- set the second point 9 bars left from the first one
        time2=temp[0];
    }
    //--- if the second point's price is not set, move it 300 points lower than the first
    if(!price2)
        price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    //--- if the third point's time is not set, it coincides with the second point's date
    if(!time3)
        time3=time2;
    //--- if the third point's price is not set, it is equal to the first point's one
    if(!price3)
        price3=price1;
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
        InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
        InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)

```



```

    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing ellipse anchor points' coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the ellipse
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
//--- create an ellipse
    if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
        InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the ellipse anchor points
//--- loop counter
    int v_steps=accuracy/5;

```



```

//--- move the first and second anchor points
for(int i=0;i<v_steps;i++)
{
    //--- use the following values
    if(p1<accuracy-1)
        p1+=1;
    if(p2>1)
        p2-=1;
    //--- shift the points
    if(!EllipsePointChange(0,InpName,0,date[d1],price[p1]))
        return;
    if(!EllipsePointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
}
//--- 1 second of delay
Sleep(1000);
//--- loop counter
int h_steps=bars/5;
//--- move the third anchor point
for(int i=0;i<h_steps;i++)
{
    //--- use the following value
    if(d3>1)
        d3-=1;
    //--- move the point
    if(!EllipsePointChange(0,InpName,2,date[d3],price[p3]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- delete ellipse from the chart
EllipseDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}

```


OBJ_ARROW_THUMB_UP

Thumbs Up sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Thumbs Up\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ThumbUp";      // Sign name
input int         InpDate=75;              // Anchor point date in %
input int         InpPrice=25;            // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color       InpColor=clrRed;        // Sign color
```



```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;    // Border line style
input int               InpWidth=5;            // Sign size
input bool              InpBack=false;         // Background sign
input bool              InpSelection=true;      // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;           // Priority for mouse click
//+-----+
//| Create Thumbs Up sign                                     |
//+-----+
bool ArrowThumbUpCreate(const long          chart_ID=0,          // chart's ID
                        const string        name="ThumbUp",      // sign name
                        const int           sub_window=0,        // subwindow index
                        datetime            time=0,              // anchor point
                        double              price=0,             // anchor point
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                        const color         clr=clrRed,          // sign color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                        const int           width=3,             // sign size
                        const bool          back=false,          // in the background
                        const bool          selection=true,       // highlight to move
                        const bool          hidden=true,          // hidden in the object list
                        const long          z_order=0)            // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Thumbs Up\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool ArrowThumbUpMove(const long   chart_ID=0,      // chart's ID
                     const string name="ThumbUp",  // object name
                     datetime     time=0,          // anchor point time coordinate
                     double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Thumbs Up sign anchor type |
//+-----+
bool ArrowThumbUpAnchorChange(const long   chart_ID=0,      // chart's ID
                             const string name="ThumbUp",  // object name
                             const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
    ResetLastError();
//--- change anchor type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ",GetLastError());
        return(false);
    }
}

```



```

//--- successful execution
    return(true);
}

//+-----+
//| Delete Thumbs Up sign |
//+-----+

bool ArrowThumbUpDelete(const long   chart_ID=0,      // chart's ID
                        const string name="ThumbUp") // sign name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Thumbs Up\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+

void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

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

void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;

```



```

//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create Thumbs Up sign on the chart
    if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
    int h_steps=bars/4;
//--- move the anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d>1)
            d-=1;
        //--- move the point
        if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
    }

```



```
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    int v_steps=accuracy/4;
    //--- move the anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p<accuracy-1)
            p+=1;
        //--- move the point
        if(!ArrowThumbUpMove(0, InpName, date[d], price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- change anchor point location relative to the sign
    ArrowThumbUpAnchorChange(0, InpName, ANCHOR_BOTTOM);
    //--- redraw the chart
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the sign from the chart
    ArrowThumbUpDelete(0, InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}
```


OBJ_ARROW_THUMB_DOWN

Thumbs Down sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Thumbs Down\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ThumbDown";      // Sign name
input int         InpDate=25;                // Anchor point date in %
input int         InpPrice=75;               // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
```



```

input color      InpColor=clrRed;           // Sign color
input ENUM_LINE_STYLE InpStyle=STYLE_DOT;    // Border line style
input int        InpWidth=5;                // Sign size
input bool       InpBack=false;             // Background sign
input bool       InpSelection=true;         // Highlight to move
input bool       InpHidden=true;           // Hidden in the object list
input long       InpZOrder=0;              // Priority for mouse click

//+-----+
//| Create Thumbs Down sign |
//+-----+

bool ArrowThumbDownCreate(const long      chart_ID=0,           // chart's ID
                          const string    name="ThumbDown",     // sign name
                          const int       sub_window=0,         // subwindow ID
                          datetime        time=0,              // anchor point
                          double          price=0,              // anchor price
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                          const color      clr=clrRed,          // sign color
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                          const int        width=3,             // sign size
                          const bool        back=false,         // in the background
                          const bool        selection=true,     // highlight to move
                          const bool        hidden=true,        // hidden in the object list
                          const long        z_order=0)          // priority for mouse click
{
    //--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
    //--- reset the error value
    ResetLastError();
    //--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Thumbs Down\" sign! Error code = ",GetLastError());
        return(false);
    }
    //--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the sign by mouse
    //--- when creating a graphical object using ObjectCreate function, the object cannot
    //--- highlighted and moved by default. Inside this method, selection parameter
    //--- is true by default making it possible to highlight and move the object

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool ArrowThumbDownMove(const long   chart_ID=0,      // chart's ID
                        const string name="ThumbDown", // object name
                        datetime     time=0,           // anchor point time coordinate
                        double        price=0)          // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Thumbs Down sign anchor type |
//+-----+
bool ArrowThumbDownAnchorChange(const long   chart_ID=0,      // chart's ID
                                const string name="ThumbDown", // object name
                                const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
    ResetLastError();
//--- change anchor type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ",GetLastError());
        return(false);
    }
}

```



```

    }
    //--- successful execution
    return(true);
}
//+-----+
//| Delete Thumbs Down sign |
//+-----+
bool ArrowThumbDownDelete(const long   chart_ID=0,      // chart's ID
                          const string name="ThumbDown") // sign name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Thumbs Down\" sign! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
    //--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
    //--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size

```



```

    int accuracy=1000;
    //--- arrays for storing the date and price values to be used
    //--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
    //--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
    //--- fill the array of prices
    //--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
    //--- create Thumbs Down sign on the chart
    if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
    //--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //--- now, move the anchor point and change its position relative to the sign
    //--- loop counter
    int h_steps=bars/4;
    //--- move the anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d<bars-1)
            d+=1;
        //--- move the point
        if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())

```



```

        return;

        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- loop counter
    int v_steps=accuracy/4;
    //--- move the anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p>1)
            p-=1;
        //--- move the point
        if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
    }
    //--- change anchor point location relative to the sign
    ArrowThumbDownAnchorChange(0,InpName,ANCHOR_TOP);
    //--- redraw the chart
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the sign from the chart
    ArrowThumbDownDelete(0,InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}

```


OBJ_ARROW_UP

Arrow Up sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Arrow Up\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ArrowUp";      // Sign name
input int         InpDate=25;              // Anchor point date in %
input int         InpPrice=25;             // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color       InpColor=clrRed;        // Sign color
```



```

input ENUM_LINE_STYLE  InpStyle=STYLE_DOT;    // Border line style
input int               InpWidth=5;           // Sign size
input bool              InpBack=false;         // Background sign
input bool              InpSelection=false;    // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+-----+
//| Create Arrow Up sign                                     |
//+-----+
bool ArrowUpCreate(const long      chart_ID=0,        // chart's ID
                   const string   name="ArrowUp",    // sign name
                   const int      sub_window=0,      // subwindow index
                   datetime       time=0,            // anchor point time
                   double         price=0,           // anchor point price
                   const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                   const color    clr=clrRed,       // sign color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                   const int      width=3,          // sign size
                   const bool     back=false,       // in the background
                   const bool     selection=true,   // highlight to move
                   const bool     hidden=true,      // hidden in the object list
                   const long      z_order=0)        // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Arrow Up\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool ArrowUpMove(const long   chart_ID=0,      // chart's ID
                 const string name="ArrowUp", // object name
                 datetime    time=0,          // anchor point time coordinate
                 double       price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Arrow Down sign anchor type |
//+-----+
bool ArrowUpAnchorChange(const long   chart_ID=0,      // chart's ID
                        const string name="ArrowUp",   // object name
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
    ResetLastError();
//--- change anchor point location
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ",GetLastError());
        return(false);
    }
}

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Delete Arrow Up sign |
//+-----+
bool ArrowUpDelete(const long   chart_ID=0,      // chart's ID
                   const string name="ArrowUp") // sign name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Arrow Up\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;

```



```

//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Up sign on the chart
    if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
    int v_steps=accuracy/2;
//--- move the anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p<accuracy-1)
            p+=1;
        //--- move the point
        if(!ArrowUpMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
    }

```



```
        //--- redraw the chart
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- change anchor point location relative to the sign
    ArrowUpAnchorChange(0, InpName, ANCHOR_BOTTOM);
    //--- redraw the chart
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the sign from the chart
    ArrowUpDelete(0, InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}
```


OBJ_ARROW_DOWN

Arrow Down sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Arrow Down\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ArrowDown";      // Sign name
input int         InpDate=75;                // Anchor point date in %
input int         InpPrice=75;               // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color       InpColor=clrRed;           // Sign color
input ENUM_LINE_STYLE InpStyle=STYLE_DOT;    // Border line style
```



```

input int      InpWidth=5;           // Sign size
input bool     InpBack=false;        // Background sign
input bool     InpSelection=false;    // Highlight to move
input bool     InpHidden=true;       // Hidden in the object list
input long     InpZOrder=0;          // Priority for mouse click

//+-----+
//| Create Arrow Down sign                                     |
//+-----+
bool ArrowDownCreate(const long      chart_ID=0,           // chart's ID
                    const string     name="ArrowDown",     // sign name
                    const int        sub_window=0,        // subwindow index
                    datetime         time=0,              // anchor point time
                    double            price=0,            // anchor point price
                    const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                    const color      clr=clrRed,         // sign color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                    const int        width=3,            // sign size
                    const bool       back=false,         // in the background
                    const bool       selection=true,     // highlight to move
                    const bool       hidden=true,        // hidden in the object list
                    const long       z_order=0)          // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Arrow Down\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

```



```

//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}
//+-----+
//| Move the anchor point |
//+-----+
bool ArrowDownMove(const long   chart_ID=0,      // chart's ID
                   const string name="ArrowDown", // object name
                   datetime     time=0,          // anchor point time coordinate
                   double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}
//+-----+
//| Change Arrow Down sign anchor type |
//+-----+
bool ArrowDownAnchorChange(const long   chart_ID=0,      // chart's ID
                           const string name="ArrowDown", // object name
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
ResetLastError();
//--- change anchor point location
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": failed to change anchor type! Error code = ",GetLastError());
    return(false);
}
//--- successful execution

```



```

    return(true);
}

//+-----+
//| Delete Arrow Down sign |
//+-----+
bool ArrowDownDelete(const long   chart_ID=0,      // chart's ID
                     const string name="ArrowDown") // sign name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Arrow Down\" sign! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
    //--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
    //--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- price array size
    int accuracy=1000;
    //--- arrays for storing the date and price values to be used

```



```

//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Down sign on the chart
    if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
    int v_steps=accuracy/2;
//--- move the anchor point
    for(int i=0;i<v_steps;i++)
    {
        //--- use the following value
        if(p>1)
            p-=1;
        //--- move the point
        if(!ArrowDownMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
    }

```



```
        ChartRedraw();
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- change anchor point location relative to the sign
    ArrowDownAnchorChange(0, InpName, ANCHOR_TOP);
    //--- redraw the chart
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the sign from the chart
    ArrowDownDelete(0, InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}
```


OBJ_ARROW_STOP

Stop sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Stop sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Stop\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ArrowStop";      // Sign name
input int         InpDate=10;               // Anchor point date in %
input int         InpPrice=50;              // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color       InpColor=clrRed;          // Sign color
```



```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;           // Border line style
input int                InpWidth=5;                  // Sign size
input bool               InpBack=false;               // Background sign
input bool               InpSelection=false;           // Highlight to move
input bool               InpHidden=true;              // Hidden in the object list
input long               InpZOrder=0;                 // Priority for mouse click
//+-----+
//| Create Stop sign                                     |
//+-----+
bool ArrowStopCreate(const long      chart_ID=0,       // chart's ID
                    const string    name="ArrowStop", // sign name
                    const int       sub_window=0,     // subwindow index
                    datetime         time=0,           // anchor point time
                    double           price=0,          // anchor point price
                    const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                    const color      clr=clrRed,       // sign color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                    const int        width=3,         // sign size
                    const bool        back=false,     // in the background
                    const bool        selection=true,  // highlight to move
                    const bool        hidden=true,    // hidden in the object list
                    const long        z_order=0)      // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Stop\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool ArrowStopMove(const long   chart_ID=0,      // chart's ID
                  const string name="ArrowStop", // object name
                  datetime     time=0,          // anchor point time coordinate
                  double        price=0)        // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Stop sign anchor type |
//+-----+
bool ArrowStopAnchorChange(const long   chart_ID=0,      // chart's ID
                          const string  name="ArrowStop", // object name
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor point
{
//--- reset the error value
    ResetLastError();
//--- change anchor type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ",GetLastError());
        return(false);
    }
}

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Delete Stop sign |
//+-----+
bool ArrowStopDelete(const long   chart_ID=0,      // chart's ID
                     const string name="ArrowStop") // label name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Stop\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;

```



```

//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create Stop sign on the chart
    if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
    int h_steps=bars*2/5;
//--- move the anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d<bars-1)
            d+=1;
        //--- move the point
        if(!ArrowStopMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
    }

```



```

    //--- redraw the chart
    ChartRedraw();
    // 0.025 seconds of delay
    Sleep(25);
}
//--- change anchor point location relative to the sign
ArrowStopAnchorChange(0, InpName, ANCHOR_TOP);
//--- redraw the chart
ChartRedraw();
//--- loop counter
h_steps=bars*2/5;
//--- move the anchor point
for(int i=0; i<h_steps; i++)
{
    //--- use the following value
    if(d<bars-1)
        d+=1;
    //--- move the point
    if(!ArrowStopMove(0, InpName, date[d], price[p]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.025 seconds of delay
    Sleep(25);
}
//--- 1 second of delay
Sleep(1000);
//--- delete the sign from the chart
ArrowStopDelete(0, InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}

```


OBJ_ARROW_CHECK

Check sign.



Note

Anchor point position relative to the sign can be selected from [ENUM_ARROW_ANCHOR](#) enumeration.

Large signs (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

Example

The following script creates and moves Check sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Check\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="ArrowCheck"; // Sign name
input int         InpDate=10;           // Anchor point date in %
input int         InpPrice=50;          // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color       InpColor=clrRed;      // Sign color
```



```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;    // Border line style
input int               InpWidth=5;            // Sign size
input bool              InpBack=false;         // Background sign
input bool              InpSelection=false;     // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;           // Priority for mouse click
//+-----+
//| Create Check sign                                     |
//+-----+
bool ArrowCheckCreate(const long      chart_ID=0,        // chart's ID
                     const string    name="ArrowCheck", // sign name
                     const int       sub_window=0,      // subwindow index
                     datetime         time=0,           // anchor point time
                     double           price=0,          // anchor point price
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                     const color      clr=clrRed,       // sign color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                     const int        width=3,          // sign size
                     const bool       back=false,       // in the background
                     const bool       selection=true,    // highlight to move
                     const bool       hidden=true,       // hidden in the object list
                     const long       z_order=0)        // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create the sign
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Check\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool ArrowCheckMove(const long   chart_ID=0,      // chart's ID
                    const string name="ArrowCheck", // object name
                    datetime    time=0,          // anchor point time coordinate
                    double       price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change Check anchor type |
//+-----+
bool ArrowCheckAnchorChange(const long   chart_ID=0,      // chart's ID
                            const string name="ArrowCheck", // object name
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
    ResetLastError();
//--- change anchor type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ",GetLastError());
        return(false);
    }
}

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Delete Check sign |
//+-----+
bool ArrowCheckDelete(const long   chart_ID=0,          // chart's ID
                     const string name="ArrowCheck") // sign name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Check\" sign! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;

```



```

//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the sign
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create Check sign on the chart
    if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
    int h_steps=bars*2/5;
//--- move the anchor point
    for(int i=0;i<h_steps;i++)
    {
        //--- use the following value
        if(d<bars-1)
            d+=1;
        //--- move the point
        if(!ArrowCheckMove(0,InpName,date[d],price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
    }

```



```

        //--- redraw the chart
        ChartRedraw();
        // 0.025 seconds of delay
        Sleep(25);
    }
    //--- change anchor point location relative to the sign
    ArrowCheckAnchorChange(0, InpName, ANCHOR_BOTTOM);
    //--- redraw the chart
    ChartRedraw();
    //--- loop counter
    h_steps=bars*2/5;
    //--- move the anchor point
    for(int i=0; i<h_steps; i++)
    {
        //--- use the following value
        if(d<bars-1)
            d+=1;
        //--- move the point
        if(!ArrowCheckMove(0, InpName, date[d], price[p]))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.025 seconds of delay
        Sleep(25);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the sign from the chart
    ArrowCheckDelete(0, InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}

```


OBJ_ARROW_LEFT_PRICE

Left Price Label



Example

The following script creates and moves left price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the left price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="LeftPrice";    // Price label name
input int         InpDate=100;            // Anchor point date in %
input int         InpPrice=10;            // Anchor point price in %
input color       InpColor=clrRed;        // Price label color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Border line style
input int         InpWidth=2;             // Price label size
input bool        InpBack=false;          // Background label
input bool        InpSelection=true;      // Highlight to move
input bool        InpHidden=true;        // Hidden in the object list
input long        InpZOrder=0;           // Priority for mouse click
//+-----+
//| Create the left price label |
```



```

//+-----+
bool ArrowLeftPriceCreate(const long      chart_ID=0,      // chart's ID
                        const string     name="LeftPrice", // price label name
                        const int        sub_window=0,     // subwindow index
                        datetime          time=0,          // anchor point time
                        double            price=0,          // anchor point price
                        const color       clr=clrRed,       // price label color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                        const int        width=1,          // price label size
                        const bool        back=false,       // in the background
                        const bool        selection=true,    // highlight to move
                        const bool        hidden=true,       // hidden in the object list
                        const long        z_order=0)         // priority for moving
{
    //--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
    //--- reset the error value
    ResetLastError();
    //--- create a price label
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create the left price label! Error code = ",GetLastError());
        return(false);
    }
    //--- set the label color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set the label size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the label by mouse
    //--- when creating a graphical object using ObjectCreate function, the object cannot
    //--- highlighted and moved by default. Inside this method, selection parameter
    //--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
    //--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
    //--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
    //--- successful execution
    return(true);
}
//+-----+
//| Move the anchor point |
//+-----+

```



```

bool ArrowLeftPriceMove(const long   chart_ID=0,      // chart's ID
                        const string name="LeftPrice", // label name
                        datetime     time=0,          // anchor point time coordinate
                        double        price=0)         // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Delete the left price label from the chart |
//+-----+
bool ArrowLeftPriceDelete(const long   chart_ID=0,      // chart's ID
                           const string name="LeftPrice") // label name
{
//--- reset the error value
ResetLastError();
//--- delete the label
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": failed to delete the left price label! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
if(!time)
    time=TimeCurrent();

```



```

//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the label
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create the left price label on the chart
    if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
}

```



```
    }  
    //--- redraw the chart and wait for 1 second  
    ChartRedraw();  
    Sleep(1000);  
    //--- now, move the anchor point  
    //--- loop counter  
    int v_steps=accuracy*4/5;  
    //--- move the anchor point  
    for(int i=0;i<v_steps;i++)  
    {  
        //--- use the following value  
        if(p<accuracy-1)  
            p+=1;  
        //--- move the point  
        if(!ArrowLeftPriceMove(0,InpName,date[d],price[p]))  
            return;  
        //--- check if the script's operation has been forcefully disabled  
        if(IsStopped())  
            return;  
        //--- redraw the chart  
        ChartRedraw();  
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the label from the chart  
    ArrowLeftPriceDelete(0,InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_ARROW_RIGHT_PRICE

Right Price Label.



Example

The following script creates and moves right price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the right price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="RightPrice"; // Price label name
input int         InpDate=0;           // Anchor point date in %
input int         InpPrice=90;         // Anchor point price in %
input color       InpColor=clrRed;     // Price label color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Border line style
input int         InpWidth=2;          // Price label size
input bool        InpBack=false;       // Background label
input bool        InpSelection=true;    // Highlight to move
input bool        InpHidden=true;      // Hidden in the object list
input long        InpZOrder=0;         // Priority for mouse click
//+-----+
//| Create the right price label |
```



```

//+-----+
bool ArrowRightPriceCreate(const long      chart_ID=0,      // chart's ID
                          const string    name="RightPrice", // price label name
                          const int       sub_window=0,     // subwindow index
                          datetime        time=0,           // anchor point time
                          double          price=0,          // anchor point price
                          const color      clr=clrRed,      // price label color
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                          const int       width=1,          // price label size
                          const bool       back=false,       // in the background
                          const bool       selection=true,   // highlight to mouse
                          const bool       hidden=true,      // hidden in the chart
                          const long      z_order=0)         // priority for mouse click
{
    //--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
    //--- reset the error value
    ResetLastError();
    //--- create a price label
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create the right price label! Error code = ",GetLastError());
        return(false);
    }
    //--- set the label color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set the label size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the label by mouse
    //--- when creating a graphical object using ObjectCreate function, the object cannot
    //--- highlighted and moved by default. Inside this method, selection parameter
    //--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
    //--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
    //--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
    //--- successful execution
    return(true);
}
//+-----+
//| Move the anchor point |
//+-----+

```



```

bool ArrowRightPriceMove(const long   chart_ID=0,          // chart's ID
                        const string name="RightPrice",    // label name
                        datetime     time=0,              // anchor point time coordinat
                        double        price=0)            // anchor point price coordinat
{
//--- if point position is not set, move it to the current bar having Bid price
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
ResetLastError();
//--- move the anchor point
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": failed to move the anchor point! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Delete the right price label from the chart |
//+-----+
bool ArrowRightPriceDelete(const long   chart_ID=0,          // chart's ID
                           const string name="RightPrice") // label name
{
//--- reset the error value
ResetLastError();
//--- delete the label
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": failed to delete the right price label! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
if(!time)
    time=TimeCurrent();

```



```

//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }

//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }

//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the label
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create the right price label on the chart
    if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
        InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
}

```



```
    }  
    //--- redraw the chart and wait for 1 second  
    ChartRedraw();  
    Sleep(1000);  
    //--- now, move the anchor point  
    //--- loop counter  
    int v_steps=accuracy*4/5;  
    //--- move the anchor point  
    for(int i=0;i<v_steps;i++)  
    {  
        //--- use the following value  
        if(p>1)  
            p-=1;  
        //--- move the point  
        if(!ArrowRightPriceMove(0,InpName,date[d],price[p]))  
            return;  
        //--- check if the script's operation has been forcefully disabled  
        if(IsStopped())  
            return;  
        //--- redraw the chart  
        ChartRedraw();  
    }  
    //--- 1 second of delay  
    Sleep(1000);  
    //--- delete the label from the chart  
    ArrowRightPriceDelete(0,InpName);  
    ChartRedraw();  
    //--- 1 second of delay  
    Sleep(1000);  
    //---  
}
```


OBJ_ARROW_BUY

Buy sign.



Example

The following script creates and moves Buy sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Buy\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'3,95,172'; // Color of signs
//+-----+
//| Create Buy sign |
//+-----+

bool ArrowBuyCreate(const long      chart_ID=0,      // chart's ID
                    const string   name="ArrowBuy",  // sign name
                    const int      sub_window=0,     // subwindow index
                    datetime        time=0,          // anchor point time
                    double          price=0,         // anchor point price
                    const color     clr=C'3,95,172', // sign color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style (when highl
                    const int      width=1,         // line size (when highl
                    const bool      back=false,     // in the background
                    const bool      selection=false, // highlight to move
```



```

        const bool        hidden=true,        // hidden in the object list
        const long        z_order=0)        // priority for mouse click

    {
//--- set anchor point coordinates if they are not set
        ChangeArrowEmptyPoint(time,price);
//--- reset the error value
        ResetLastError();
//--- create the sign
        if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
        {
            Print(__FUNCTION__,
                ": failed to create \"Buy\" sign! Error code = ",GetLastError());
            return(false);
        }
//--- set a sign color
        ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
        ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
        ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
        ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
        ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
        ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
        ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
        ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
        return(true);
    }
//+-----+
//| Move the anchor point |
//+-----+
bool ArrowBuyMove(const long    chart_ID=0,        // chart's ID
                  const string name="ArrowBuy",    // object name
                  datetime      time=0,            // anchor point time coordinate
                  double         price=0)           // anchor point price coordinate
    {
//--- if point position is not set, move it to the current bar having Bid price
        if(!time)
            time=TimeCurrent();
        if(!price)
            price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
        ResetLastError();
//--- move the anchor point
        if(!ObjectMove(chart_ID,name,0,time,price))

```



```

    {
        Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete Buy sign |
//+-----+
bool ArrowBuyDelete(const long   chart_ID=0,      // chart's ID
                    const string name="ArrowBuy") // sign name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete \"Buy\" sign! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // array for storing dates of visible bars
    double   low[];  // array for storing Low prices of visible bars
    double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

```



```

//--- memory allocation
ArrayResize(date,bars);
ArrayResize(low,bars);
ArrayResize(high,bars);
//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of Low prices
if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
{
    Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
    return;
}
//--- fill the array of High prices
if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
{
    Print("Failed to copy the values of High prices! Error code = ",GetLastError());
    return;
}
//--- create Buy signs in Low point for each visible bar
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- move Buy signs to High point for each visible bar
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}

```



```
//--- delete Buy signs
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//---
}
```


OBJ_ARROW_SELL

Sell sign.



Example

The following script creates and moves Sell sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Sell\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'225,68,29'; // Color of signs
//+-----+
//| Create Sell sign |
//+-----+

bool ArrowSellCreate(const long      chart_ID=0,          // chart's ID
                    const string    name="ArrowSell",     // sign name
                    const int       sub_window=0,         // subwindow index
                    const datetime   time=0,              // anchor point time
                    const double     price=0,              // anchor point price
                    const color      clr=C'225,68,29',    // sign color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style (when high
                    const int        width=1,              // line size (when high
                    const bool       back=false,           // in the background
                    const bool       selection=false,      // highlight to move
```



```

        const bool      hidden=true,          // hidden in the object
        const long      z_order=0)           // priority for mouse click

    {
//--- set anchor point coordinates if they are not set
        ChangeArrowEmptyPoint(time,price);
//--- reset the error value
        ResetLastError();
//--- create the sign
        if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
        {
            Print(__FUNCTION__,
                ": failed to create \"Sell\" sign! Error code = ",GetLastError());
            return(false);
        }
//--- set a sign color
        ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
        ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
        ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
        ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
        ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
        ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
        ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
        ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
        return(true);
    }
//+-----+
//| Move the anchor point |
//+-----+
bool ArrowSellMove(const long   chart_ID=0,      // chart's ID
                  const string name="ArrowSell", // object name
                  datetime     time=0,          // anchor point time coordinate
                  double        price=0)        // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))

```



```

    {
        Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete Sell sign |
//+-----+
bool ArrowSellDelete(const long   chart_ID=0,      // chart's ID
                    const string name="ArrowSell") // sign name
{
//--- reset the error value
    ResetLastError();
//--- delete the sign
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete \"Sell\" sign! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // array for storing dates of visible bars
    double   low[];  // array for storing Low prices of visible bars
    double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

```



```

//--- memory allocation
ArrayResize(date,bars);
ArrayResize(low,bars);
ArrayResize(high,bars);
//--- fill the array of dates
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Failed to copy time values! Error code = ",GetLastError());
    return;
}
//--- fill the array of Low prices
if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
{
    Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
    return;
}
//--- fill the array of High prices
if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
{
    Print("Failed to copy the values of High prices! Error code = ",GetLastError());
    return;
}
//--- create Sell signs in High point for each visible bar
for(int i=0;i<bars;i++)
{
    if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpColor))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//--- move Sell signs to Low point for each visible bar
for(int i=0;i<bars;i++)
{
    if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}

```



```
//--- delete Sell signs
for(int i=0;i<bars;i++)
{
    if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
        return;
    //--- redraw the chart
    ChartRedraw();
    // 0.05 seconds of delay
    Sleep(50);
}
//---
}
```


OBJ_ARROW

Arrow object.



Note

Anchor point position relative to the object can be selected from [ENUM_ARROW_ANCHOR](#).

Large arrows (more than 5) can only be created by setting the appropriate [OBJPROP_WIDTH](#) property value when writing a code in MetaEditor.

The necessary arrow type can be selected by setting one of the [Wingdings](#) font's symbol codes.

Example

The following script creates Arrow object on the chart and changes its type. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates a random arrow in the chart window."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Arrow";           // Arrow name
input int         InpDate=50;                 // Anchor point date in %
input int         InpPrice=50;                // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
```



```

input color      InpColor=clrDodgerBlue; // Arrow color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Border line style
input int        InpWidth=10;           // Arrow size
input bool       InpBack=false;         // Background arrow
input bool       InpSelection=false;    // Highlight to move
input bool       InpHidden=true;        // Hidden in the object list
input long       InpZOrder=0;           // Priority for mouse click
//+-----+
//| Create the arrow |
//+-----+
bool ArrowCreate(const long      chart_ID=0,           // chart's ID
                 const string   name="Arrow",         // arrow name
                 const int      sub_window=0,         // subwindow index
                 datetime        time=0,              // anchor point time
                 double          price=0,             // anchor point price
                 const uchar     arrow_code=252,       // arrow code
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor point position
                 const color     clr=clrRed,          // arrow color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // border line style
                 const int       width=3,            // arrow size
                 const bool      back=false,         // in the background
                 const bool      selection=true,     // highlight to move
                 const bool      hidden=true,        // hidden in the object list
                 const long      z_order=0)           // priority for mouse click
{
    //--- set anchor point coordinates if they are not set
    ChangeArrowEmptyPoint(time,price);
    //--- reset the error value
    ResetLastError();
    //--- create an arrow
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create an arrow! Error code = ",GetLastError());
        return(false);
    }
    //--- set the arrow code
    ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
    //--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- set the arrow color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- set the border line style
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- set the arrow's size
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the arrow by mouse

```



```

//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+

bool ArrowMove(const long   chart_ID=0,    // chart's ID
               const string name="Arrow",  // object name
               datetime     time=0,        // anchor point time coordinate
               double        price=0)      // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change the arrow code |
//+-----+

bool ArrowCodeChange(const long   chart_ID=0,    // chart's ID
                     const string name="Arrow",  // object name
                     const uchar  code=252)     // arrow code
{
//--- reset the error value
    ResetLastError();
//--- change the arrow code
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))
    {

```



```

        Print(__FUNCTION__,
              ": failed to change the arrow code! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Change anchor type                                     |
//+-----+
bool ArrowAnchorChange(const long      chart_ID=0,      // chart's ID
                      const string    name="Arrow",    // object name
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
{
//--- reset the error value
    ResetLastError();
//--- change anchor type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
    {
        Print(__FUNCTION__,
              ": failed to change anchor type! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete an arrow                                       |
//+-----+
bool ArrowDelete(const long  chart_ID=0,  // chart's ID
                 const string name="Arrow") // arrow name
{
//--- reset the error value
    ResetLastError();
//--- delete an arrow
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete an arrow! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values     |
//| for empty ones                                       |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)

```



```

{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
    int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
    datetime date[];
    double price[];
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- define points for drawing the arrow
    int d=InpDate*(bars-1)/100;
    int p=InpPrice*(accuracy-1)/100;
//--- create an arrow on the chart

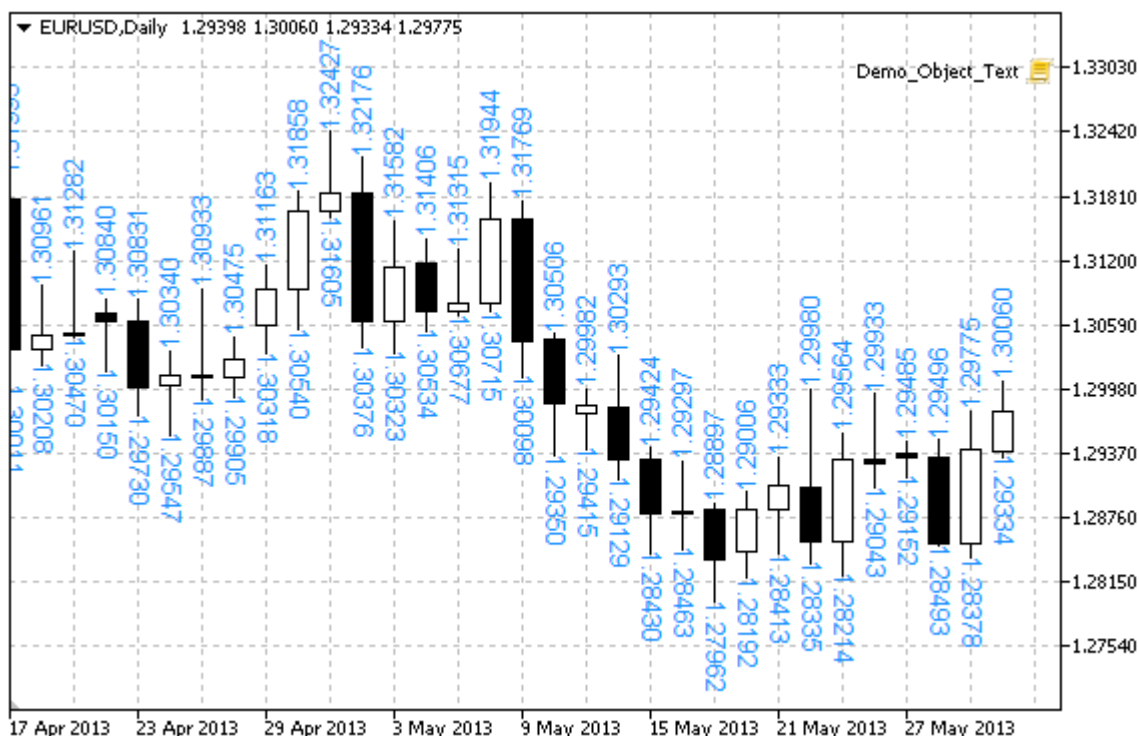
```



```
    if(!ArrowCreate(0, InpName, 0, date[d], price[p], 32, InpAnchor, InpColor,
        InpStyle, InpWidth, InpBack, InpSelection, InpHidden, InpZOrder))
    {
        return;
    }
//--- redraw the chart
    ChartRedraw();
//--- consider all cases of creating arrows in the loop
    for(int i=33; i<256; i++)
    {
        if(!ArrowCodeChange(0, InpName, (uchar)i))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // half a second of delay
        Sleep(500);
    }
//--- 1 second of delay
    Sleep(1000);
//--- delete the arrow from the chart
    ArrowDelete(0, InpName);
    ChartRedraw();
//--- 1 second of delay
    Sleep(1000);
//---
}
```


OBJ_TEXT

Text object.



Note

Anchor point position relative to the text can be selected from [ENUM_ANCHOR_POINT](#) enumeration. You can also change text slope angle using [OBJPROP_ANGLE](#) property.

Example

The following script creates several Text objects on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Text\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpFont="Arial";           // Font
input int         InpFontSize=10;            // Font size
input color       InpColor=clrRed;           // Color
input double      InpAngle=90.0;             // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_BOTTOM; // Anchor type
input bool        InpBack=false;             // Background object
input bool        InpSelection=false;        // Highlight to move
input bool        InpHidden=true;           // Hidden in the object list
input long        InpZOrder=0;              // Priority for mouse click
```



```

//+-----+
//| Creating Text object |
//+-----+
bool TextCreate(const long      chart_ID=0,          // chart's ID
                const string    name="Text",         // object name
                const int       sub_window=0,        // subwindow index
                datetime         time=0,              // anchor point time
                double           price=0,              // anchor point price
                const string     text="Text",         // the text itself
                const string     font="Arial",        // font
                const int        font_size=10,        // font size
                const color      clr=clrRed,          // color
                const double     angle=0.0,           // text slope
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor type
                const bool       back=false,          // in the background
                const bool       selection=false,     // highlight to move
                const bool       hidden=true,         // hidden in the object list
                const long       z_order=0)           // priority for mouse click
{
    //--- set anchor point coordinates if they are not set
    ChangeTextEmptyPoint(time,price);
    //--- reset the error value
    ResetLastError();
    //--- create Text object
    if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create \"Text\" object! Error code = ",GetLastError());
        return(false);
    }
    //--- set the text
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
    //--- set text font
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
    //--- set font size
    ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
    //--- set the slope angle of the text
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
    //--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- enable (true) or disable (false) the mode of moving the object by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
    //--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
}

```



```

//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the anchor point |
//+-----+
bool TextMove(const long   chart_ID=0, // chart's ID
              const string name="Text", // object name
              datetime     time=0,      // anchor point time coordinate
              double       price=0)     // anchor point price coordinate
{
//--- if point position is not set, move it to the current bar having Bid price
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
              ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change the object text |
//+-----+
bool TextChange(const long   chart_ID=0, // chart's ID
                const string name="Text", // object name
                const string text="Text") // text
{
//--- reset the error value
    ResetLastError();
//--- change object text
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": failed to change the text! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

```



```

//+-----+
//| Delete Text object |
//+-----+
bool TextDelete(const long chart_ID=0, // chart's ID
                const string name="Text") // object name
{
//--- reset the error value
    ResetLastError();
//--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Text\" object! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeTextEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // array for storing dates of visible bars
    double low[]; // array for storing Low prices of visible bars
    double high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(low,bars);
    ArrayResize(high,bars);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {

```



```

        Print("Failed to copy time values! Error code = ", GetLastError());
        return;
    }
    ///--- fill the array of Low prices
    if(CopyLow(Symbol(), Period(), 0, bars, low)==-1)
    {
        Print("Failed to copy the values of Low prices! Error code = ", GetLastError());
        return;
    }
    ///--- fill the array of High prices
    if(CopyHigh(Symbol(), Period(), 0, bars, high)==-1)
    {
        Print("Failed to copy the values of High prices! Error code = ", GetLastError());
        return;
    }
    ///--- define how often texts are to be displayed
    int scale=(int)ChartGetInteger(0, CHART_SCALE);
    ///--- define the step
    int step=1;
    switch(scale)
    {
        case 0:
            step=12;
            break;
        case 1:
            step=6;
            break;
        case 2:
            step=4;
            break;
        case 3:
            step=2;
            break;
    }
    ///--- create texts for High and Low bars' values (with gaps)
    for(int i=0; i<bars; i+=step)
    {
        ///--- create the texts
        if(!TextCreate(0, "TextHigh_" + (string)i, 0, date[i], high[i], DoubleToString(high[i],
            InpColor, InpAngle, InpAnchor, InpBack, InpSelection, InpHidden, InpZOrder))
        {
            return;
        }
        if(!TextCreate(0, "TextLow_" + (string)i, 0, date[i], low[i], DoubleToString(low[i], 5),
            InpColor, -InpAngle, InpAnchor, InpBack, InpSelection, InpHidden, InpZOrder))
        {
            return;
        }
    }
    ///--- check if the script's operation has been forcefully disabled

```



```
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- half a second of delay
    Sleep(500);
    //--- delete the texts
    for(int i=0;i<bars;i+=step)
    {
        if(!TextDelete(0,"TextHigh_"+(string)i))
            return;
        if(!TextDelete(0,"TextLow_"+(string)i))
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //---
}
```


OBJ_LABEL

Label object.



Note

Anchor point position relative to the label can be selected from [ENUM_ANCHOR_POINT](#) enumeration. Anchor point coordinates are set in pixels.

You can also select text label anchoring corner from [ENUM_BASE_CORNER](#) enumeration.

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Label\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Label";           // Label name
input int         InpX=150;                  // X-axis distance
input int         InpY=150;                  // Y-axis distance
input string      InpFont="Arial";           // Font
input int         InpFontSize=14;            // Font size
input color       InpColor=clrRed;           // Color
input double      InpAngle=0.0;              // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Anchor type
```



```

input bool      InpBack=false;           // Background object
input bool      InpSelection=true;       // Highlight to move
input bool      InpHidden=true;         // Hidden in the object list
input long      InpZOrder=0;            // Priority for mouse click
//+-----+
//| Create a text label |
//+-----+
bool LabelCreate(const long      chart_ID=0,           // chart's ID
                 const string    name="Label",        // label name
                 const int       sub_window=0,        // subwindow index
                 const int       x=0,                 // X coordinate
                 const int       y=0,                 // Y coordinate
                 const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner for
                 const string    text="Label",        // text
                 const string    font="Arial",        // font
                 const int       font_size=10,        // font size
                 const color      clr=clrRed,         // color
                 const double     angle=0.0,          // text slope
                 const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor type
                 const bool       back=false,         // in the background
                 const bool       selection=false,    // highlight to move
                 const bool       hidden=true,        // hidden in the object list
                 const long       z_order=0)          // priority for mouse click
{
//--- reset the error value
    ResetLastError();
//--- create a text label
    if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": failed to create text label! Error code = ",GetLastError());
        return(false);
    }
//--- set label coordinates
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set the chart's corner, relative to which point coordinates are defined
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
    ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the slope angle of the text
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set color

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}
//+-----+
//| Move the text label |
//+-----+
bool LabelMove(const long   chart_ID=0,    // chart's ID
               const string name="Label",  // label name
               const int    x=0,          // X coordinate
               const int    y=0)          // Y coordinate
{
//--- reset the error value
    ResetLastError();
//--- move the text label
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
              ": failed to move X coordinate of the label! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
              ": failed to move Y coordinate of the label! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Change corner of the chart for binding the label |
//+-----+
bool LabelChangeCorner(const long   chart_ID=0,    // chart's ID
                      const string name="Label",  // label name
                      const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // chart corner
{
//--- reset the error value
    ResetLastError();
//--- change anchor corner

```



```

    if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
    {
        Print(__FUNCTION__,
              ": failed to change the anchor corner! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change the label text |
//+-----+
bool LabelTextChange(const long   chart_ID=0,    // chart's ID
                    const string name="Label",  // object name
                    const string text="Text")   // text
{
//--- reset the error value
    ResetLastError();
//--- change object text
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": failed to change the text! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete a text label |
//+-----+
bool LabelDelete(const long   chart_ID=0,    // chart's ID
                const string name="Label")  // label name
{
//--- reset the error value
    ResetLastError();
//--- delete the label
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete a text label! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

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

```



```

void OnStart()
{
    //--- store the label's coordinates in the local variables
    int x=InpX;
    int y=InpY;
    //--- chart window size
    long x_distance;
    long y_distance;
    //--- set window size
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Failed to get the chart width! Error code = ",GetLastError());
        return;
    }
    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Failed to get the chart height! Error code = ",GetLastError());
        return;
    }
    //--- check correctness of the input parameters
    if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
    //--- prepare initial text for the label
    string text;
    StringConcatenate(text,"Upper left corner: ",x,"",y);
    //--- create a text label on the chart
    if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,InpFontSize,
        InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
    //--- redraw the chart and wait for half a second
    ChartRedraw();
    Sleep(500);
    //--- move the label and change its text simultaneously
    //--- number of iterations by axes
    int h_steps=(int)(x_distance/2-InpX);
    int v_steps=(int)(y_distance/2-InpY);
    //--- move the label down
    for(int i=0;i<v_steps;i++)
    {
        //--- change the coordinate
        y+=2;
        //--- move the label and change its text
        MoveAndTextChange(x,y,"Upper left corner: ");
    }
}

```



```

//--- half a second of delay
Sleep(500);
//--- move the label to the right
for(int i=0;i<h_steps;i++)
{
    //--- change the coordinate
    x+=2;
    //--- move the label and change its text
    MoveAndTextChange(x,y,"Upper left corner: ");
}
//--- half a second of delay
Sleep(500);
//--- move the label up
for(int i=0;i<v_steps;i++)
{
    //--- change the coordinate
    y-=2;
    //--- move the label and change its text
    MoveAndTextChange(x,y,"Upper left corner: ");
}
//--- half a second of delay
Sleep(500);
//--- move the label to the left
for(int i=0;i<h_steps;i++)
{
    //--- change the coordinate
    x-=2;
    //--- move the label and change its text
    MoveAndTextChange(x,y,"Upper left corner: ");
}
//--- half a second of delay
Sleep(500);
//--- now, move the point by changing the anchor corner
//--- move to the lower left corner
if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
    return;
//--- change the label text
StringConcatenate(text,"Lower left corner: ",x,"",y);
if(!LabelTextChange(0,InpName,text))
    return;
//--- redraw the chart and wait for two seconds
ChartRedraw();
Sleep(2000);
//--- move to the lower right corner
if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
    return;
//--- change the label text
StringConcatenate(text,"Lower right corner: ",x,"",y);
if(!LabelTextChange(0,InpName,text))

```



```

        return;
//--- redraw the chart and wait for two seconds
        ChartRedraw();
        Sleep(2000);
//--- move to the upper right corner
        if(!LabelChangeCorner(0, InpName, CORNER_RIGHT_UPPER))
            return;
//--- change the label text
        StringConcatenate(text, "Upper right corner: ", x, ", ", y);
        if(!LabelTextChange(0, InpName, text))
            return;
//--- redraw the chart and wait for two seconds
        ChartRedraw();
        Sleep(2000);
//--- move to the upper left corner
        if(!LabelChangeCorner(0, InpName, CORNER_LEFT_UPPER))
            return;
//--- change the label text
        StringConcatenate(text, "Upper left corner: ", x, ", ", y);
        if(!LabelTextChange(0, InpName, text))
            return;
//--- redraw the chart and wait for two seconds
        ChartRedraw();
        Sleep(2000);
//--- delete the label
        LabelDelete(0, InpName);
//--- redraw the chart and wait for half a second
        ChartRedraw();
        Sleep(500);
//---
    }
//+-----+
//| The function moves the object and changes its text |
//+-----+
bool MoveAndTextChange(const int x, const int y, string text)
{
//--- move the label
    if(!LabelMove(0, InpName, x, y))
        return(false);
//--- change the label text
    StringConcatenate(text, text, x, ", ", y);
    if(!LabelTextChange(0, InpName, text))
        return(false);
//--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return(false);
//--- redraw the chart
    ChartRedraw();
// 0.01 seconds of delay

```



```
Sleep(10);  
//--- exit the function  
return(true);  
}
```


OBJ_BUTTON

Button object.



Note

Anchor point coordinates are set in pixels. You can select button anchoring corner from [ENUM_BASE_CORNER](#).

Example

The following script creates and moves Button object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the button on the chart."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Button";           // Button name
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input string      InpFont="Arial";           // Font
input int         InpFontSize=14;            // Font size
input color       InpColor=clrBlack;         // Text color
input color       InpBackColor=C'236,233,216'; // Background color
input color       InpBorderColor=clrNONE;    // Border color
input bool        InpState=false;            // Pressed/Released
input bool        InpBack=false;            // Background object
input bool        InpSelection=false;        // Highlight to move
```



```

input bool      InpHidden=true;           // Hidden in the object list
input long      InpZOrder=0;              // Priority for mouse click
//+-----+
//| Create the button                      |
//+-----+
bool ButtonCreate(const long      chart_ID=0,           // chart's ID
                  const string    name="Button",       // button name
                  const int       sub_window=0,        // subwindow index
                  const int       x=0,                 // X coordinate
                  const int       y=0,                 // Y coordinate
                  const int       width=50,            // button width
                  const int       height=18,           // button height
                  const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner for button
                  const string    text="Button",       // text
                  const string    font="Arial",        // font
                  const int       font_size=10,        // font size
                  const color     clr=clrBlack,        // text color
                  const color     back_clr=C'236,233,216', // background color
                  const color     border_clr=clrNONE,  // border color
                  const bool       state=false,        // pressed/released
                  const bool       back=false,         // in the background
                  const bool       selection=false,    // highlight to mouse
                  const bool       hidden=true,        // hidden in the chart
                  const long       z_order=0)           // priority for mouse click
{
//--- reset the error value
ResetLastError();
//--- create the button
if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
{
Print(__FUNCTION__,
      ": failed to create the button! Error code = ",GetLastError());
return(false);
}
//--- set button coordinates
ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set button size
ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the chart's corner, relative to which point coordinates are defined
ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set text color

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
    ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
    ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- set button state
    ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- enable (true) or disable (false) the mode of moving the button by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move the button |
//+-----+
bool ButtonMove(const long   chart_ID=0,    // chart's ID
                const string name="Button", // button name
                const int    x=0,           // X coordinate
                const int    y=0)           // Y coordinate
{
//--- reset the error value
    ResetLastError();
//--- move the button
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
              ": failed to move X coordinate of the button! Error code = ",GetLastError()
        );
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
              ": failed to move Y coordinate of the button! Error code = ",GetLastError()
        );
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change button size |
//+-----+
bool ButtonChangeSize(const long   chart_ID=0,    // chart's ID

```



```

        const string name="Button", // button name
        const int   width=50,       // button width
        const int   height=18)      // button height

    {
//--- reset the error value
    ResetLastError();
//--- change the button size
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
            ": failed to change the button width! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
            ": failed to change the button height! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change corner of the chart for binding the button |
//+-----+
bool ButtonChangeCorner(const long      chart_ID=0,          // chart's ID
                        const string    name="Button",       // button name
                        const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // chart corner

{
//--- reset the error value
    ResetLastError();
//--- change anchor corner
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
    {
        Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change button text |
//+-----+
bool ButtonTextChange(const long      chart_ID=0,          // chart's ID
                      const string    name="Button",       // button name
                      const string    text="Text")         // text

{
//--- reset the error value

```



```

    ResetLastError();
//--- change object text
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": failed to change the text! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete the button |
//+-----+
bool ButtonDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Button") // button name
{
//--- reset the error value
    ResetLastError();
//--- delete the button
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete the button! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- chart window size
    long x_distance;
    long y_distance;
//--- set window size
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Failed to get the chart width! Error code = ",GetLastError());
        return;
    }
    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Failed to get the chart height! Error code = ",GetLastError());
        return;
    }
//--- define the step for changing the button size

```



```

int x_step=(int)x_distance/32;
int y_step=(int)y_distance/32;
//--- set the button coordinates and its size
int x=(int)x_distance/32;
int y=(int)y_distance/32;
int x_size=(int)x_distance*15/16;
int y_size=(int)y_distance*15/16;
//--- create the button
if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFont,InpFontSize,
    InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,InpHidden,Inp
    {
        return;
    }
//--- redraw the chart
ChartRedraw();
//--- reduce the button in the loop
int i=0;
while(i<13)
{
    //--- half a second of delay
    Sleep(500);
    //--- switch the button to the pressed state
    ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
    //--- redraw the chart and wait for 0.2 second
    ChartRedraw();
    Sleep(200);
    //--- redefine coordinates and button size
    x+=x_step;
    y+=y_step;
    x_size-=x_step*2;
    y_size-=y_step*2;
    //--- reduce the button
    ButtonMove(0,InpName,x,y);
    ButtonChangeSize(0,InpName,x_size,y_size);
    //--- bring the button back to the released state
    ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
    //--- redraw the chart
    ChartRedraw();
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- increase the loop counter
    i++;
}
//--- half a second of delay
Sleep(500);
//--- delete the button
ButtonDelete(0,InpName);
ChartRedraw();

```



```
//--- wait for 1 second  
    Sleep(1000);  
//---  
}
```


OBJ_CHART

Chart object.



Note

Anchor point coordinates are set in pixels. You can select anchoring corner from [ENUM_BASE_CORNER](#) enumeration.

Symbol, period and scale can be selected for Chart object. Price scale and date display mode can also be enabled/disabled.

Example

The following script creates and moves Chart object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Chart\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Chart";           // Object name
input string      InpSymbol="EURUSD";        // Symbol
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H1;    // Period
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Anchoring corner
input int         InpScale=2;                 // Scale
input bool        InpDateScale=true;          // Time scale display
input bool        InpPriceScale=true;         // Price scale display
```



```

input color      InpColor=clrRed;           // Border color when highlighted
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style when highlighted
input int        InpPointWidth=1;          // Point size to move
input bool       InpBack=false;            // Background object
input bool       InpSelection=true;        // Highlight to move
input bool       InpHidden=true;          // Hidden in the object list
input long       InpZOrder=0;             // Priority for mouse click

//+-----+
//| Creating Chart object |
//+-----+

bool ObjectChartCreate(const long      chart_ID=0,           // chart's ID
                       const string   name="Chart",         // object name
                       const int      sub_window=0,         // subwindow
                       const string   symbol="EURUSD",       // symbol
                       const ENUM_TIMEFRAMES period=PERIOD_H1, // period
                       const int      x=0,                 // X coordinate
                       const int      y=0,                 // Y coordinate
                       const int      width=300,           // width
                       const int      height=200,          // height
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // anchoring
                       const int      scale=2,             // scale
                       const bool      date_scale=true,     // time scale
                       const bool      price_scale=true,    // price scale
                       const color     clr=clrRed,          // border color
                       const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                       const int      point_width=1,        // move point
                       const bool      back=false,          // in the background
                       const bool      selection=false,     // highlight
                       const bool      hidden=true,         // hidden in the list
                       const long      z_order=0)           // priority

{
//--- reset the error value
ResetLastError();

//--- create Chart object
if(!ObjectCreate(chart_ID,name,OBJ_CHART,sub_window,0,0))
{
Print(__FUNCTION__,
      ": failed to create \"Chart\" object! Error code = ",GetLastError());
return(false);
}

//--- set object coordinates
ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);

//--- set object size
ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);

//--- set the chart's corner, relative to which point coordinates are defined
ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);

//--- set the symbol

```



```

    ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol);
//--- set the period
    ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period);
//--- set the scale
    ObjectSetInteger(chart_ID,name,OBJPROP_CHART_SCALE,scale);
//--- display (true) or hide (false) the time scale
    ObjectSetInteger(chart_ID,name,OBJPROP_DATE_SCALE,date_scale);
//--- display (true) or hide (false) the price scale
    ObjectSetInteger(chart_ID,name,OBJPROP_PRICE_SCALE,price_scale);
//--- set the border color when object highlighting mode is enabled
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Sets the symbol and time frame of the Chart object |
//+-----+
bool ObjectChartSetSymbolAndPeriod(const long      chart_ID=0,      // chart's
                                   const string     name="Chart",    // object name
                                   const string     symbol="EURUSD",  // symbol
                                   const ENUM_TIMEFRAMES period=PERIOD_H1) // time frame
{
//--- reset the error value
    ResetLastError();
//--- set Chart object's symbol and time frame
    if(!ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol))
    {
        Print(__FUNCTION__,
              ": failed to set a symbol for \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period))
    {
        Print(__FUNCTION__,
              ": failed to set a period for \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
}

```



```

    }
    //--- successful execution
    return(true);
}

//+-----+
//| Move Chart object |
//+-----+

bool ObjectChartMove(const long   chart_ID=0,    // chart's ID (not Chart object's one)
                    const string name="Chart",  // object name
                    const int    x=0,          // X coordinate
                    const int    y=0)          // Y coordinate
{
    //--- reset the error value
    ResetLastError();
    //--- move the object
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
              ": failed to move X coordinate of \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
              ": failed to move Y coordinate of \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Change Chart object size |
//+-----+

bool ObjectChartChangeSize(const long   chart_ID=0,    // chart's ID (not Chart object's one)
                          const string name="Chart",  // object name
                          const int    width=300,     // width
                          const int    height=200)    // height
{
    //--- reset the error value
    ResetLastError();
    //--- change the object size
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
              ": failed to change the width of \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {

```



```

        Print(__FUNCTION__,
              ": failed to change the height of \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Return Chart object's ID |
//+-----+
long ObjectChartGetID(const long   chart_ID=0,    // chart's ID (not Chart object's one)
                     const string name="Chart") // object name
{
//--- prepare the variable to get Chart object's ID
    long id=-1;
//--- reset the error value
    ResetLastError();
//--- get ID
    if(!ObjectGetInteger(chart_ID,name,OBJPROP_CHART_ID,0,id))
    {
        Print(__FUNCTION__,
              ": failed to get \"Chart\" object's ID! Error code = ",GetLastError());
    }
//--- return the result
    return(id);
}
//+-----+
//| Delete Chart object |
//+-----+
bool ObjectChartDelete(const long   chart_ID=0,    // chart's ID (not Chart object's one)
                      const string name="Chart") // object name
{
//--- reset the error value
    ResetLastError();
//--- delete the button
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Chart\" object! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{

```



```

//--- get the number of symbols in Market Watch
int symbols=SymbolsTotal(true);
//--- check if the symbol with a specified name is present in the symbol list
bool exist=false;
for(int i=0;i<symbols;i++)
    if(InpSymbol==SymbolName(i,true))
    {
        exist=true;
        break;
    }
if(!exist)
{
    Print("Error! ",InpSymbol," symbol is not present in \"Market Watch\"!");
    return;
}
//--- check validity of input parameters
if(InpScale<0 || InpScale>5)
{
    Print("Error! Incorrect values of input parameters!");
    return;
}

//--- chart window size
long x_distance;
long y_distance;
//--- set window size
if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
{
    Print("Failed to get the chart width! Error code = ",GetLastError());
    return;
}
if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
{
    Print("Failed to get the chart height! Error code = ",GetLastError());
    return;
}

//--- set Chart object coordinates and its size
int x=(int)x_distance/16;
int y=(int)y_distance/16;
int x_size=(int)x_distance*7/16;
int y_size=(int)y_distance*7/16;
//--- create Chart object
if(!ObjectChartCreate(0,InpName,0,InpSymbol,InpPeriod,x,y,x_size,y_size,InpCorner,1,
    InpPriceScale,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
    {
        return;
    }
}

//--- redraw the chart and wait for 1 second
ChartRedraw();

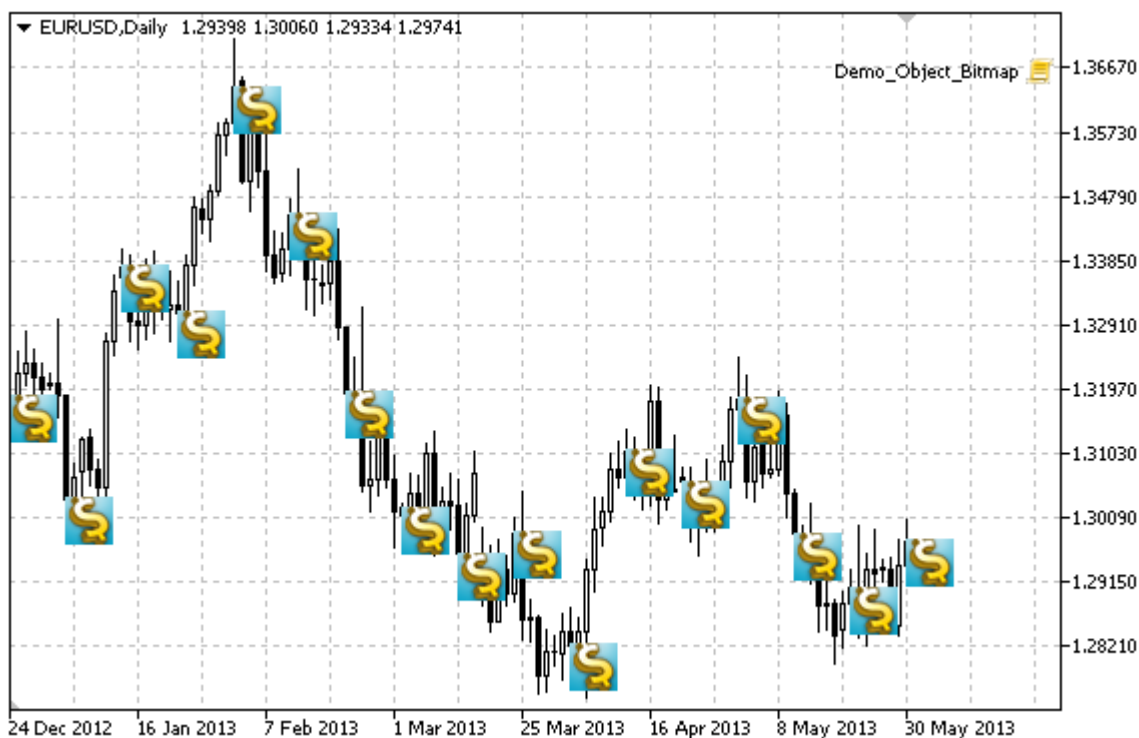
```



```
Sleep(1000);
//--- stretch Chart object
int steps=(int)MathMin(x_distance*7/16,y_distance*7/16);
for(int i=0;i<steps;i++)
{
    //--- resize
    x_size+=1;
    y_size+=1;
    if(!ObjectChartChangeSize(0,InpName,x_size,y_size))
        return;
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart and wait for 0.01 seconds
    ChartRedraw();
    Sleep(10);
}
//--- half a second of delay
Sleep(500);
//--- change chart's time frame
if(!ObjectChartSetSymbolAndPeriod(0,InpName,InpSymbol,PERIOD_M1))
    return;
ChartRedraw();
//--- three seconds of delay
Sleep(3000);
//--- delete the object
ObjectChartDelete(0,InpName);
ChartRedraw();
//--- wait for 1 second
Sleep(1000);
//---
}
```


OBJ_BITMAP

Bitmap object.



Note

For Bitmap object, you can select [visibility scope](#) of an image.

Example

The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates a bitmap in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpFile="\\Images\\dollar.bmp"; // Bitmap file name
input int         InpWidth=24;                  // Visibility scope X coordinate
input int         InpHeight=24;                 // Visibility scope Y coordinate
input int         InpXOffset=4;                 // Visibility scope shift by X axis
input int         InpYOffset=4;                 // Visibility scope shift by Y axis
input color       InpColor=clrRed;              // Border color when highlighted
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;     // Line style when highlighted
input int         InpPointWidth=1;             // Point size to move
input bool        InpBack=false;               // Background object
input bool        InpSelection=false;          // Highlight to move
```



```

input bool      InpHidden=true;           // Hidden in the object list
input long      InpZOrder=0;              // Priority for mouse click
//+-----+
//| Create a bitmap in the chart window |
//+-----+
bool BitmapCreate(const long      chart_ID=0,      // chart's ID
                  const string    name="Bitmap",  // bitmap name
                  const int       sub_window=0,    // subwindow index
                  datetime         time=0,         // anchor point time
                  double           price=0,        // anchor point price
                  const string     file="",        // bitmap file name
                  const int        width=10,      // visibility scope X coord
                  const int        height=10,     // visibility scope Y coord
                  const int        x_offset=0,    // visibility scope shift k
                  const int        y_offset=0,    // visibility scope shift k
                  const color      clr=clrRed,    // border color when highlight
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style when highlight
                  const int        point_width=1, // move point size
                  const bool       back=false,    // in the background
                  const bool       selection=false, // highlight to move
                  const bool       hidden=true,   // hidden in the object list
                  const long       z_order=0)     // priority for mouse click
{
//--- set anchor point coordinates if they are not set
    ChangeBitmapEmptyPoint(time,price);
//--- reset the error value
    ResetLastError();
//--- create a bitmap
    if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": failed to create a bitmap in the chart window! Error code = ",GetLastError());
        return(false);
    }
//--- set the path to the image file
    if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
    {
        Print(__FUNCTION__,
              ": failed to load the image! Error code = ",GetLastError());
        return(false);
    }
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility scope
//--- the default part is the upper left area of an image; the values allow

```



```

//--- performing a shift from this area displaying another part of the image
ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- set the border color when object highlighting mode is enabled
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
return(true);
}

//+-----+
//| Set a new image for the bitmap |
//+-----+
bool BitmapSetImage(const long   chart_ID=0,    // chart's ID
                    const string name="Bitmap", // bitmap name
                    const string file="")       // path to the file
{
//--- reset the error value
ResetLastError();
//--- set the path to the image file
if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
{
    Print(__FUNCTION__,
          ": failed to load the image! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Move a bitmap in the chart window |
//+-----+
bool BitmapMove(const long   chart_ID=0,    // chart's ID
                const string name="Bitmap", // bitmap name
                datetime     time=0,        // anchor point time
                double       price=0)       // anchor point price
{
//--- if point position is not set, move it to the current bar having Bid price

```



```

    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
    ResetLastError();
//--- move the anchor point
    if(!ObjectMove(chart_ID,name,0,time,price))
    {
        Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change visibility scope (bitmap) size |
//+-----+
bool BitmapChangeSize(const long   chart_ID=0,    // chart's ID
                      const string name="Bitmap", // bitmap name
                      const int    width=0,       // bitmap width
                      const int    height=0)      // bitmap height
{
//--- reset the error value
    ResetLastError();
//--- change bitmap size
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
            ": failed to change the bitmap width! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
            ": failed to change the bitmap height! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change coordinate of the upper left corner of the visibility scope |
//+-----+
bool BitmapMoveVisibleArea(const long   chart_ID=0,    // chart's ID
                           const string name="Bitmap", // bitmap name
                           const int    x_offset=0,    // visibility scope X coordinate
                           const int    y_offset=0)    // visibility scope Y coordinate

```



```

{
//--- reset the error value
    ResetLastError();
//--- change the bitmap's visibility scope coordinates
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
    {
        Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
    {
        Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Delete a bitmap |
//+-----+
bool BitmapDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Bitmap") // bitmap name
{
//--- reset the error value
    ResetLastError();
//--- delete the label
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": failed to delete a bitmap! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Check anchor point values and set default values |
//| for empty ones |
//+-----+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
{
//--- if the point's time is not set, it will be on the current bar
    if(!time)
        time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

```



```

    }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // array for storing dates of visible bars
    double   close[]; // array for storing Close prices
//--- bitmap file name
    string   file="\\Images\\dollar.bmp";
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
    ArrayResize(date,bars);
    ArrayResize(close,bars);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- fill the array of Close prices
    if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
    {
        Print("Failed to copy the values of Close prices! Error code = ",GetLastError());
        return;
    }
//--- define how often the images should be displayed
    int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- define the step
    int step=1;
    switch(scale)
    {
        case 0:
            step=27;
            break;
        case 1:
            step=14;
            break;
        case 2:
            step=7;
            break;
        case 3:
            step=4;
            break;
        case 4:
            step=2;
            break;
    }
}

```



```

    }
    //--- create bitmaps for High and Low bars' values (with gaps)
    for(int i=0;i<bars;i+=step)
    {
        //--- create the bitmaps
        if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,InpWidth,InpHe
            InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
        {
            return;
        }
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //--- half a second of delay
    Sleep(500);
    //--- delete Sell signs
    for(int i=0;i<bars;i+=step)
    {
        if(!BitmapDelete(0,"Bitmap_"+(string)i))
            return;
        if(!BitmapDelete(0,"Bitmap_"+(string)i))
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.05 seconds of delay
        Sleep(50);
    }
    //---
}

```


OBJ_BITMAP_LABEL

Bitmap Label object.



Note

Anchor point position relative to the label can be selected from [ENUM_ANCHOR_POINT](#) enumeration. Anchor point coordinates are set in pixels.

You can also select bitmap anchoring corner from [ENUM_BASE_CORNER](#) enumeration.

For bitmap label, you can select [visibility scope](#) of an image.

Example

The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Bitmap Label\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="BmpLabel";           // Label name
input string      InpFileOn="//Images\\dollar.bmp"; // File name for On mode
input string      InpFileOff="//Images\\euro.bmp"; // File name for Off mode
input bool        InpState=false;               // Label pressed/released
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Anchor type
input color       InpColor=clrRed;              // Border color when highlighted
```



```

input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;           // Line style when highlight
input int               InpPointWidth=1;               // Point size to move
input bool              InpBack=false;                 // Background object
input bool              InpSelection=false;            // Highlight to move
input bool              InpHidden=true;                // Hidden in the object list
input long              InpZOrder=0;                   // Priority for mouse click

//+-----+
//| Create Bitmap Label object |
//+-----+

bool BitmapLabelCreate(const long      chart_ID=0,           // chart's ID
                      const string    name="BmpLabel",      // label name
                      const int        sub_window=0,         // subwindow
                      const int        x=0,                  // X coordinate
                      const int        y=0,                  // Y coordinate
                      const string      file_on="",           // image in On mode
                      const string      file_off="",          // image in Off mode
                      const int         width=0,              // width
                      const int         height=0,             // height
                      const int         x_offset=10,           // X offset
                      const int         y_offset=10,           // Y offset
                      const bool         state=false,          // pressed/released
                      const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner
                      const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor type
                      const color       clr=clrRed,           // border color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                      const int         point_width=1,        // move point width
                      const bool         back=false,           // background
                      const bool         selection=false,       // highlight
                      const bool         hidden=true,          // hidden in the list
                      const long         z_order=0)             // priority

{
//--- reset the error value
ResetLastError();
//--- create a bitmap label
if(!ObjectCreate(chart_ID,name,OBJ_BITMAP_LABEL,sub_window,0,0))
{
Print(__FUNCTION__,
      ": failed to create \"Bitmap Label\" object! Error code = ",GetLastError());
return(false);
}
//--- set the images for On and Off modes
if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,0,file_on))
{
Print(__FUNCTION__,
      ": failed to load the image for On mode! Error code = ",GetLastError());
return(false);
}
if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,1,file_off))
{

```



```

        Print(__FUNCTION__,
              ": failed to load the image for Off mode! Error code = ", GetLastError());
        return(false);
    }
//--- set label coordinates
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility scope
//--- the default part is the upper left area of an image; the values allow
//--- performing a shift from this area displaying another part of the image
    ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
    ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- define the label's status (pressed or released)
    ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- set the chart's corner, relative to which point coordinates are defined
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set anchor type
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set the border color when object highlighting mode is enabled
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}
//+-----+
//| Set a new image for Bitmap label object |
//+-----+
bool BitmapLabelSetImage(const long   chart_ID=0,      // chart's ID
                        const string name="BmpLabel",  // label name
                        const int    on_off=0,        // modifier (On or Off)
                        const string file="")          // path to the file

```



```

{
//--- reset the error value
    ResetLastError();
//--- set the path to the image file
    if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
    {
        Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Move Bitmap Label object |
//+-----+
bool BitmapLabelMove(const long   chart_ID=0,      // chart's ID
                    const string name="BmpLabel", // label name
                    const int    x=0,             // X coordinate
                    const int    y=0)             // Y coordinate
{
//--- reset the error value
    ResetLastError();
//--- move the object
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",GetLastError()
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",GetLastError()
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change visibility scope (object) size |
//+-----+
bool BitmapLabelChangeSize(const long   chart_ID=0,      // chart's ID
                          const string name="BmpLabel", // label name
                          const int    width=0,         // label width
                          const int    height=0)         // label height
{
//--- reset the error value
    ResetLastError();

```



```

//--- change the object size
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
{
    Print(__FUNCTION__,
        ": failed to change the object width! Error code = ",GetLastError());
    return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
{
    Print(__FUNCTION__,
        ": failed to change the object height! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Change coordinate of the upper left corner of the visibility scope |
//+-----+
bool BitmapLabelMoveVisibleArea(const long   chart_ID=0,      // chart's ID
                                const string name="BmpLabel", // label name
                                const int    x_offset=0,       // visibility scope X co
                                const int    y_offset=0)       // visibility scope Y co

{
    //--- reset the error value
    ResetLastError();
    //--- change the object's visibility scope coordinates
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
    {
        Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Error code = ",C
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
    {
        Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Error code = ",C
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Delete "Bitmap label" object |
//+-----+
bool BitmapLabelDelete(const long   chart_ID=0,      // chart's ID
                        const string name="BmpLabel") // label name

{
    //--- reset the error value

```



```

    ResetLastError();
//--- delete the label
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Bitmap label\" object! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- chart window size
    long x_distance;
    long y_distance;
//--- set window size
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Failed to get the chart width! Error code = ",GetLastError());
        return;
    }
    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Failed to get the chart height! Error code = ",GetLastError());
        return;
    }
//--- define bitmap label coordinates
    int x=(int)x_distance/2;
    int y=(int)y_distance/2;
//--- set label size and visibility scope coordinates
    int width=32;
    int height=32;
    int x_offset=0;
    int y_offset=0;
//--- place bitmap label at the center of the window
    if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height,x_offset,y_
        InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHid
    {
        return;
    }
//--- redraw the chart and wait one second
    ChartRedraw();
    Sleep(1000);
//--- change label's visibility scope size in the loop
    for(int i=0;i<6;i++)

```



```

    {
        //--- change visibility scope size
        width--;
        height--;
        if(!BitmapLabelChangeSize(0,InpName,width,height))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.3 seconds of delay
        Sleep(300);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- change label's visibility scope coordinates in the loop
    for(int i=0;i<2;i++)
    {
        //--- change visibility scope coordinates
        x_offset++;
        y_offset++;
        if(!BitmapLabelMoveVisibleArea(0,InpName,x_offset,y_offset))
            return;
        //--- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //--- redraw the chart
        ChartRedraw();
        // 0.3 seconds of delay
        Sleep(300);
    }
    //--- 1 second of delay
    Sleep(1000);
    //--- delete the label
    BitmapLabelDelete(0,InpName);
    ChartRedraw();
    //--- 1 second of delay
    Sleep(1000);
    //---
}

```


OBJ_EDIT

Edit object.



Note

Anchor point coordinates are set in pixels. You can select Edit anchoring corner from [ENUM_BASE_CORNER](#) enumeration.

You can also select one of the text alignment types inside Edit from [ENUM_ALIGN_MODE](#) enumeration.

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Edit\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Edit";           // Object name
input string      InpText="Text";           // Object text
input string      InpFont="Arial";          // Font
input int         InpFontSize=14;           // Font size
input ENUM_ALIGN_MODE InpAlign=ALIGN_CENTER; // Text alignment type
input bool        InpReadOnly=false;        // Permission to edit
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input color       InpColor=clrBlack;        // Text color
```



```

input color      InpBackColor=clrWhite;      // Background color
input color      InpBorderColor=clrBlack;    // Border color
input bool       InpBack=false;              // Background object
input bool       InpSelection=false;         // Highlight to move
input bool       InpHidden=true;             // Hidden in the object list
input long       InpZOrder=0;                // Priority for mouse click
//+-----+
//| Create Edit object |
//+-----+
bool EditCreate(const long      chart_ID=0,      // chart's ID
                const string    name="Edit",    // object name
                const int       sub_window=0,    // subwindow index
                const int       x=0,             // X coordinate
                const int       y=0,             // Y coordinate
                const int       width=50,        // width
                const int       height=18,       // height
                const string     text="Text",    // text
                const string     font="Arial",   // font
                const int       font_size=10,    // font size
                const ENUM_ALIGN_MODE align=ALIGN_CENTER, // alignment type
                const bool       read_only=false, // ability to edit
                const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner for a
                const color      clr=clrBlack,   // text color
                const color      back_clr=clrWhite, // background color
                const color      border_clr=clrNONE, // border color
                const bool       back=false,     // in the background
                const bool       selection=false, // highlight to move
                const bool       hidden=true,    // hidden in the object list
                const long       z_order=0)      // priority for mouse click
{
//--- reset the error value
    ResetLastError();
//--- create edit field
    if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": failed to create \"Edit\" object! Error code = ",GetLastError());
        return(false);
    }
//--- set object coordinates
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set object size
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the text
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);

```



```

//--- set font size
    ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the type of text alignment in the object
    ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- enable (true) or cancel (false) read-only mode
    ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,read_only);
//--- set the chart's corner, relative to which object coordinates are defined
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set text color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
    ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
    ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move Edit object |
//+-----+

bool EditMove(const long   chart_ID=0, // chart's ID
              const string name="Edit", // object name
              const int    x=0,         // X coordinate
              const int    y=0)         // Y coordinate
{
//--- reset the error value
    ResetLastError();
//--- move the object
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
              ": failed to move X coordinate of the object! Error code = ",GetLastError()
        );
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
              ": failed to move Y coordinate of the object! Error code = ",GetLastError()
        );
        return(false);
    }
}

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Resize Edit object |
//+-----+
bool EditChangeSize(const long   chart_ID=0, // chart's ID
                    const string name="Edit", // object name
                    const int   width=0,     // width
                    const int   height=0)    // height
{
//--- reset the error value
    ResetLastError();
//--- change the object size
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
              ": failed to change the object width! Error code = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
              ": failed to change the object height! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Change Edit object's text |
//+-----+
bool EditTextChange(const long   chart_ID=0, // chart's ID
                    const string name="Edit", // object name
                    const string text="Text") // text
{
//--- reset the error value
    ResetLastError();
//--- change object text
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": failed to change the text! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+

```



```

//| Return Edit object text |
//+-----+
bool EditTextGet(string      &text,          // text
                 const long  chart_ID=0,    // chart's ID
                 const string name="Edit") // object name
{
//--- reset the error value
    ResetLastError();
//--- get object text
    if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
    {
        Print(__FUNCTION__,
              ": failed to get the text! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Delete Edit object |
//+-----+
bool EditDelete(const long  chart_ID=0, // chart's ID
               const string name="Edit") // object name
{
//--- reset the error value
    ResetLastError();
//--- delete the label
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": failed to delete \"Edit\" object! Error code = ",GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- chart window size
    long x_distance;
    long y_distance;
//--- set window size
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Failed to get the chart width! Error code = ",GetLastError());
        return;
    }

```



```

    }
    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Failed to get the chart height! Error code = ",GetLastError());
        return;
    }
    //-- define the step for changing the edit field
    int x_step=(int)x_distance/64;
    //-- set edit field coordinates and its size
    int x=(int)x_distance/8;
    int y=(int)y_distance/2;
    int x_size=(int)x_distance/8;
    int y_size=InpFontSize*2;
    //-- store the text in the local variable
    string text=InpText;
    //-- create edit field
    if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSize,InpAlign,1,
        InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,InpHidden,Inp
    {
        return;
    }
    //-- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
    //-- stretch the edit field
    while(x_size-x<x_distance*5/8)
    {
        //-- increase edit field's width
        x_size+=x_step;
        if(!EditChangeSize(0,InpName,x_size,y_size))
            return;
        //-- check if the script's operation has been forcefully disabled
        if(IsStopped())
            return;
        //-- redraw the chart and wait for 0.05 seconds
        ChartRedraw();
        Sleep(50);
    }
    //-- half a second of delay
    Sleep(500);
    //-- change the text
    for(int i=0;i<20;i++)
    {
        //-- add "+" at the beginning and at the end
        text="+"+text+" ";
        if(!EditTextChange(0,InpName,text))
            return;
        //-- check if the script's operation has been forcefully disabled
        if(IsStopped())

```



```
        return;

        //--- redraw the chart and wait for 0.1 seconds
        ChartRedraw();
        Sleep(100);
    }

    //--- half a second of delay
    Sleep(500);

    //--- delete edit field
    EditDelete(0, InpName);
    ChartRedraw();

    //--- wait for 1 second
    Sleep(1000);

    //---
}
```


OBJ_EVENT

Event object.



Note

When hovering mouse over the event, its text appears.

Example

The following script creates and moves Event object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Event\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="Event";      // Event name
input int         InpDate=25;           // Event date, %
input string      InpText="Text";       // Event text
input color       InpColor=clrRed;      // Event color
input int         InpWidth=1;           // Point size when highlighted
input bool        InpBack=false;        // Background event
input bool        InpSelection=false;   // Highlight to move
input bool        InpHidden=true;       // Hidden in the object list
```



```

input long          InpZOrder=0;          // Priority for mouse click
//+-----+
//| Create Event object on the chart      |
//+-----+
bool EventCreate(const long          chart_ID=0,      // chart's ID
                 const string       name="Event",    // event name
                 const int          sub_window=0,    // subwindow index
                 const string       text="Text",      // event text
                 datetime            time=0,         // time
                 const color        clr=clrRed,      // color
                 const int          width=1,         // point width when highlighted
                 const bool         back=false,      // in the background
                 const bool         selection=false, // highlight to move
                 const bool         hidden=true,     // hidden in the object list
                 const long         z_order=0)       // priority for mouse click
{
//--- if time is not set, create the object on the last bar
    if(!time)
        time=TimeCurrent();
//--- reset the error value
    ResetLastError();
//--- create Event object
    if(!ObjectCreate(chart_ID,name,OBJ_EVENT,sub_window,time,0))
    {
        Print(__FUNCTION__,
              ": failed to create \"Event\" object! Error code = ",GetLastError());
        return(false);
    }
//--- set event text
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set color
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set anchor point width if the object is highlighted
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving event by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}
//+-----+
//| Change Event object text              |
//+-----+

```



```

bool EventTextChange(const long   chart_ID=0,    // chart's ID
                    const string name="Event",  // event name
                    const string text="Text")   // text
{
    //--- reset the error value
    ResetLastError();
    //--- change object text
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": failed to change the text! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Move Event object |
//+-----+
bool EventMove(const long   chart_ID=0,    // chart's ID
               const string name="Event", // event name
               datetime      time=0)       // time
{
    //--- if time is not set, move event to the last bar
    if(!time)
        time=TimeCurrent();
    //--- reset the error value
    ResetLastError();
    //--- move the object
    if(!ObjectMove(chart_ID,name,0,time,0))
    {
        Print(__FUNCTION__,
              ": failed to move \"Event\" object! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Delete Event object |
//+-----+
bool EventDelete(const long   chart_ID=0,    // chart's ID
                 const string name="Event") // event name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the object
    if(!ObjectDelete(chart_ID,name))
    {

```



```

        Print(__FUNCTION__,
              ": failed to delete \"Event\" object! Error code = ", GetLastError());
        return(false);
    }
//--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- check correctness of the input parameters
    if(InpDate<0 || InpDate>100)
    {
        Print("Error! Incorrect values of input parameters!");
        return;
    }
//--- number of visible bars in the chart window
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing Event object anchor point's coordinates
    datetime date[];
//--- memory allocation
    ArrayResize(date,bars);
//--- fill the array of dates
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Failed to copy time values! Error code = ",GetLastError());
        return;
    }
//--- define the points to create an object
    int d=InpDate*(bars-1)/100;
//--- create Event object
    if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
                  InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- redraw the chart and wait for 1 second
    ChartRedraw();
    Sleep(1000);
//--- now, move the object
//--- loop counter
    int h_steps=bars/2;
//--- move the object
    for(int i=0;i<h_steps;i++)
    {

```



```
//--- use the following value
if(d<bars-1)
    d+=1;
//--- move the point
if(!EventMove(0,InpName,date[d]))
    return;
//--- check if the script's operation has been forcefully disabled
if(IsStopped())
    return;
//--- redraw the chart
ChartRedraw();
// 0.05 seconds of delay
Sleep(50);
}
//--- 1 second of delay
Sleep(1000);
//--- delete the channel from the chart
EventDelete(0,InpName);
ChartRedraw();
//--- 1 second of delay
Sleep(1000);
//---
}
```


OBJ_RECTANGLE_LABEL

Rectangle Label object.



Note

Anchor point coordinates are set in pixels. You can select rectangle label's anchoring corner from [ENUM_BASE_CORNER](#) enumeration. Rectangle label's border type can be selected from [ENUM_BORDER_TYPE](#) enumeration.

The object is used to create and design the custom graphical interface.

Example

The following script creates and moves Rectangle Label object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Rectangle Label\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string      InpName="RectLabel";           // Label name
input color       InpBackColor=clrSkyBlue;       // Background color
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;    // Border type
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input color       InpColor=clrDarkBlue;         // Flat border color (Flat)
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;      // Flat border style (Flat)
input int         InpLineWidth=3;               // Flat border width (Flat)
```



```

input bool      InpBack=false;           // Background object
input bool      InpSelection=true;       // Highlight to move
input bool      InpHidden=true;         // Hidden in the object list
input long      InpZOrder=0;            // Priority for mouse click
//+-----+
//| Create rectangle label |
//+-----+
bool RectLabelCreate(const long      chart_ID=0,           // chart's ID
                    const string    name="RectLabel",     // label name
                    const int       sub_window=0,         // subwindow index
                    const int       x=0,                  // X coordinate
                    const int       y=0,                  // Y coordinate
                    const int       width=50,             // width
                    const int       height=18,            // height
                    const color     back_clr=C'236,233,216', // background color
                    const ENUM_BORDER_TYPE border=BORDER_SUNKEN, // border type
                    const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner
                    const color     clr=clrRed,           // flat border color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // flat border style
                    const int       line_width=1,         // flat border width
                    const bool      back=false,           // in the background
                    const bool      selection=false,      // highlight to move
                    const bool      hidden=true,          // hidden in the object list
                    const long      z_order=0)            // priority for mouse click
{
//--- reset the error value
    ResetLastError();
//--- create a rectangle label
    if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": failed to create a rectangle label! Error code = ",GetLastError());
        return(false);
    }
//--- set label coordinates
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set label size
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set background color
    ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border type
    ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- set the chart's corner, relative to which point coordinates are defined
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set flat border color (in Flat mode)
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set flat border line style

```



```

    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set flat border width
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- display in the foreground (false) or background (true)
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
    return(true);
}

//+-----+
//| Move rectangle label |
//+-----+
bool RectLabelMove(const long   chart_ID=0,      // chart's ID
                  const string name="RectLabel", // label name
                  const int    x=0,             // X coordinate
                  const int    y=0)             // Y coordinate
{
//--- reset the error value
    ResetLastError();
//--- move the rectangle label
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
    {
        Print(__FUNCTION__,
              ": failed to move X coordinate of the label! Error code = ",GetLastError())
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
    {
        Print(__FUNCTION__,
              ": failed to move Y coordinate of the label! Error code = ",GetLastError())
        return(false);
    }
//--- successful execution
    return(true);
}

//+-----+
//| Change the size of the rectangle label |
//+-----+
bool RectLabelChangeSize(const long   chart_ID=0,      // chart's ID
                        const string name="RectLabel", // label name
                        const int    width=50,         // label width
                        const int    height=18)        // label height
{

```



```

//--- reset the error value
ResetLastError();
//--- change label size
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
{
    Print(__FUNCTION__,
          ": failed to change the label's width! Error code = ",GetLastError());
    return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
{
    Print(__FUNCTION__,
          ": failed to change the label's height! Error code = ",GetLastError());
    return(false);
}
//--- successful execution
return(true);
}

//+-----+
//| Change rectangle label border type |
//+-----+
bool RectLabelChangeBorderType(const long      chart_ID=0,          // chart's
                               const string     name="RectLabel",    // label name
                               const ENUM_BORDER_TYPE border=BORDER_SUNKEN) // border type
{
    //--- reset the error value
    ResetLastError();
    //--- change border type
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
    {
        Print(__FUNCTION__,
              ": failed to change the border type! Error code = ",GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}

//+-----+
//| Delete the rectangle label |
//+-----+
bool RectLabelDelete(const long chart_ID=0,          // chart's ID
                    const string name="RectLabel") // label name
{
    //--- reset the error value
    ResetLastError();
    //--- delete the label
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,

```



```

        ": failed to delete a rectangle label! Error code = ", GetLastError());
        return(false);
    }
    //--- successful execution
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- chart window size
    long x_distance;
    long y_distance;
    //--- set window size
    if(!ChartGetInteger(0, CHART_WIDTH_IN_PIXELS, 0, x_distance))
    {
        Print("Failed to get the chart width! Error code = ", GetLastError());
        return;
    }
    if(!ChartGetInteger(0, CHART_HEIGHT_IN_PIXELS, 0, y_distance))
    {
        Print("Failed to get the chart height! Error code = ", GetLastError());
        return;
    }
    //--- define rectangle label coordinates
    int x=(int)x_distance/4;
    int y=(int)y_distance/4;
    //--- set label size
    int width=(int)x_distance/4;
    int height=(int)y_distance/4;
    //--- create a rectangle label
    if(!RectLabelCreate(0, InpName, 0, x, y, width, height, InpBackColor, InpBorder, InpCorner,
        InpColor, InpStyle, InpLineWidth, InpBack, InpSelection, InpHidden, InpZOrder))
    {
        return;
    }
    //--- redraw the chart and wait one second
    ChartRedraw();
    Sleep(1000);
    //--- change the size of the rectangle label
    int steps=(int)MathMin(x_distance/4, y_distance/4);
    for(int i=0; i<steps; i++)
    {
        //--- resize
        width+=1;
        height+=1;
        if(!RectLabelChangeSize(0, InpName, width, height))
            return;
    }
}

```



```
    //--- check if the script's operation has been forcefully disabled
    if(IsStopped())
        return;
    //--- redraw the chart and wait for 0.01 seconds
    ChartRedraw();
    Sleep(10);
}
//--- 1 second of delay
Sleep(1000);
//--- change border type
if(!RectLabelChangeBorderType(0, InpName, BORDER_RAISED))
    return;
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- change border type
if(!RectLabelChangeBorderType(0, InpName, BORDER_SUNKEN))
    return;
//--- redraw the chart and wait for 1 second
ChartRedraw();
Sleep(1000);
//--- delete the label
RectLabelDelete(0, InpName);
ChartRedraw();
//--- wait for 1 second
Sleep(1000);
//---
}
```


Object Properties

Graphical objects can have various properties depending on the object type. Values of object properties are set up and received by corresponding [functions for working with graphical objects](#).

All objects used in technical analysis are bound to the time and price coordinates: trendline, channels, Fibonacci tools, etc. But there is a number of auxiliary objects intended to improve the user interface that are bound to the always visible part of a chart (main chart windows or indicator subwindows):

Object	ID	X/Y	Width/ Height	Date/Pr ice	OBJPROP_OP_CORNER	OBJPROP_OP_ANCHOR	OBJPROP_OP_ANGLE
Text	OBJ_TEXT	—	—	Yes	—	Yes	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	—	Yes	Yes	Yes
Button	OBJ_BUTTON	Yes	Yes	—	Yes	—	—
Bitmap	OBJ_BITMAP	—	Yes (read only)	Yes	—	Yes	—
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	—	Yes	Yes	—
Edit	OBJ_EDIT	Yes	Yes	—	Yes	—	—
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	—	Yes	—	—

The following designations are used in the table:

- **X/Y** - coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** - objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** - anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** - defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the [ENUM_BASE_CORNER](#) enumeration;
- **OBJPROP_ANCHOR** - defines the anchor point in object itself and can be one of the 9 values of the [ENUM_ANCHOR_POINT](#) enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** - defines the object rotation angle counterclockwise.

The functions defining the properties of graphical objects, as well as [ObjectCreate\(\)](#) and [ObjectMove\(\)](#) operations for creating and moving objects along the chart are actually used for sending commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. Visual changes in the properties of graphical objects are implemented when handling the queue of the chart events.

Thus, do not expect an immediate visual update of graphical objects after calling these functions. Generally, the graphical objects on the chart are updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use [ChartRedraw\(\)](#) function to forcefully update the graphical objects.

For functions [ObjectSetInteger\(\)](#) and [ObjectGetInteger\(\)](#)

ENUM_OBJECT_PROPERTY_INTEGER

Identifier	Description	Property Type
OBJPROP_COLOR	Color	color
OBJPROP_STYLE	Style	ENUM_LINE_STYLE
OBJPROP_WIDTH	Line thickness	int
OBJPROP_BACK	Object in the background	bool
OBJPROP_ZORDER	Priority of a graphical object for receiving events of clicking on a chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When objects are placed one atop another, only one of them with the highest priority will receive the CHARTEVENT_CLICK event.	long
OBJPROP_FILL	Fill an object with color (for OBJ_RECTANGLE , OBJ_TRIANGLE , OBJ_ELLIPSE , OBJ_CHANNEL , OBJ_STDDEVCHANNEL , OBJ_REGRESSION)	bool
OBJPROP_HIDDEN	Prohibit showing of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects". The true value allows to hide an object from the list. By default, true is set to the objects that display calendar events, trading history and to the objects created from MQL5 programs . To see such graphical objects and access their properties, click on the	bool

	"All" button in the "List of objects" window.	
OBJPROP_SELECTED	Object is selected	bool
OBJPROP_READONLY	Ability to edit text in the Edit object	bool
OBJPROP_TYPE	Object type	ENUM_OBJECT r/o
OBJPROP_TIME	Time coordinate	datetime modifier=number of anchor point
OBJPROP_SELECTABLE	Object availability	bool
OBJPROP_CREATETIME	Time of object creation	datetime r/o
OBJPROP_LEVELS	Number of levels	int
OBJPROP_LEVELCOLOR	Color of the line-level	color modifier=level number
OBJPROP_LEVELSTYLE	Style of the line-level	ENUM_LINE_STYLE modifier=level number
OBJPROP_LEVELWIDTH	Thickness of the line-level	int modifier=level number
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	ENUM_ALIGN_MODE
OBJPROP_FONTSIZE	Font size	int
OBJPROP_RAY_LEFT	Ray goes to the left	bool
OBJPROP_RAY_RIGHT	Ray goes to the right	bool
OBJPROP_RAY	A vertical line goes through all the windows of a chart	bool
OBJPROP_ELLIPSE	Showing the full ellipse of the Fibonacci Arc object (OBJ_FIBOARC)	bool
OBJPROP_ARROWCODE	Arrow code for the Arrow object	char
OBJPROP_TIMEFRAMES	Visibility of an object at timeframes	set of flags flags
OBJPROP_ANCHOR	Location of the anchor point of a graphical object	ENUM_ARROW_ANCHOR (for OBJ_ARROW), ENUM_ANCHOR_POINT (for OBJ_LABEL, OBJ_BITMAP_LABEL and OBJ_TEXT)
OBJPROP_XDISTANCE	The distance in pixels along the X axis from the binding corner (see note)	int

OBJPROP_YDISTANCE	The distance in pixels along the Y axis from the binding corner (see note)	int
OBJPROP_DIRECTION	Trend of the Gann object	ENUM_GANN_DIRECTION
OBJPROP_DEGREE	Level of the Elliott Wave Marking	ENUM_ELLIOT_WAVE_DEGREE
OBJPROP_DRAWLINES	Displaying lines for marking the Elliott Wave	bool
OBJPROP_STATE	Button state (pressed / depressed)	bool
OBJPROP_CHART_ID	ID of the "Chart" object (OBJ_CHART). It allows working with the properties of this object like with a normal chart using the functions described in Chart Operations , but there some exceptions .	long r/o
OBJPROP_XSIZE	The object's width along the X axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	int
OBJPROP_YSIZE	The object's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	int
OBJPROP_XOFFSET	The X coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	int
OBJPROP_YOFFSET	The Y coordinate of the upper left corner of the rectangular visible area in the graphical	int

	objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	
OBJPROP_PERIOD	Timeframe for the Chart object	ENUM_TIMEFRAMES
OBJPROP_DATE_SCALE	Displaying the time scale for the Chart object	bool
OBJPROP_PRICE_SCALE	Displaying the price scale for the Chart object	bool
OBJPROP_CHART_SCALE	The scale for the Chart object	int value in the range 0-5
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	color
OBJPROP_CORNER	The corner of the chart to link a graphical object	ENUM_BASE_CORNER
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	ENUM_BORDER_TYPE
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	color

When using [chart operations](#) for the "Chart" object ([OBJ_CHART](#)), the following limitations are imposed:

- It cannot be closed using [ChartClose\(\)](#);
- Symbol/period cannot be changed using the [ChartSetSymbolPeriod\(\)](#) function;
- The following properties are ineffective CHART_SCALE, CHART_BRING_TO_TOP, CHART_SHOW_DATE_SCALE and CHART_SHOW_PRICE_SCALE ([ENUM_CHART_PROPERTY_INTEGER](#)).

You can set a special mode of image display for [OBJ_BITMAP_LABEL](#) and [OBJ_BITMAP](#) objects. In this mode, only part of an original image (at which a rectangular visible area is applied) is displayed, while the rest of the image becomes invisible. The size of this area should be set using the properties OBJPROP_XSIZE and OBJPROP_YSIZE. The visible area can be "moved" only within the original image using the properties OBJPROP_XOFFSET and OBJPROP_YOFFSET.

For the fixed-sized objects: [OBJ_BUTTON](#), [OBJ_RECTANGLE_LABEL](#), [OBJ_EDIT](#) and [OBJ_CHART](#), properties OBJPROP_XDISTANCE and OBJPROP_YDISTANCE set the position of the top left point of the object relative to the chart corner (OBJPROP_CORNER), from which the X and Y coordinates will be counted in pixels.

For functions [ObjectSetDouble\(\)](#) and [ObjectGetDouble\(\)](#)

ENUM_OBJECT_PROPERTY_DOUBLE

Identifier	Description	Property Type
OBJPROP_PRICE	Price coordinate	double modifier=number of anchor point
OBJPROP_LEVELVALUE	Level value	double modifier=level number
OBJPROP_SCALE	Scale (properties of Gann objects and Fibonacci Arcs)	double
OBJPROP_ANGLE	Angle. For the objects with no angle specified, created from a program, the value is equal to EMPTY_VALUE	double
OBJPROP_DEVIATION	Deviation for the Standard Deviation Channel	double

For functions [ObjectSetString\(\)](#) and [ObjectGetString\(\)](#)

ENUM_OBJECT_PROPERTY_STRING

Identifier	Description	Property Type
OBJPROP_NAME	Object name	string
OBJPROP_TEXT	Description of the object (the text contained in the object)	string
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value to it	string
OBJPROP_LEVELTEXT	Level description	string modifier=level number
OBJPROP_FONT	Font	string
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also Resources	string modifier: 0-state ON, 1-state OFF
OBJPROP_SYMBOL	Symbol for the Chart object	string

For the OBJ_RECTANGLE_LABEL object ("Rectangle label") one of the three design modes can be set, to which the following values of ENUM_BORDER_TYPE correspond.

ENUM_BORDER_TYPE

Identifier	Description
BORDER_FLAT	Flat form
BORDER_RAISED	Prominent form
BORDER_SUNKEN	Concave form

For the OBJ_EDIT object ("Edit") and for the [ChartScreenShot\(\)](#) function, you can specify the horizontal alignment type using the values of the ENUM_ALIGN_MODE enumeration.

ENUM_ALIGN_MODE

Identifier	Description
ALIGN_LEFT	Left alignment
ALIGN_CENTER	Centered (only for the Edit object)
ALIGN_RIGHT	Right alignment

Example:

```
#define UP          "\x0431"

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
    string label_name="my_OBJ_LABEL_object";
    if(ObjectFind(0,label_name)<0)
    {
        Print("Object ",label_name," not found. Error code = ",GetLastError());
        //--- create Label object
        ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
        //--- set X coordinate
        ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
        //--- set Y coordinate
        ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
        //--- define text color
        ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
        //--- define text for object Label
        ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
        //--- define font
        ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
        //--- define font size
        ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
        //--- 45 degrees rotation clockwise
```



```
ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);  
//--- disable for mouse selecting  
ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);  
//--- draw it on the chart  
ChartRedraw(0);  
}  
}
```


Methods of Object Binding

Graphical objects Text, Label, Bitmap and Bitmap Label (OBJ_TEXT, OBJ_LABEL, OBJ_BITMAP and OBJ_BITMAP_LABEL) can have one of the 9 different ways of coordinate binding defined by the OBJPROP_ANCHOR property.

Object	ID	X/Y	Width/ Height	Date/Pr ice	OBJPR OP_CO RNER	OBJPR OP_AN CHOR	OBJPR OP_AN GLE
Text	OBJ_TEXT	—	—	Yes	—	Yes	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	—	Yes	Yes	Yes
Button	OBJ_BUTTON	Yes	Yes	—	Yes	—	—
Bitmap	OBJ_BITMAP	—	Yes (read only)	Yes	—	Yes	—
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	—	Yes	Yes	—
Edit	OBJ_EDIT	Yes	Yes	—	Yes	—	—
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	—	Yes	—	—

The following designations are used in the table:

- **X/Y** - coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** - objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** - anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** - defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the [ENUM_BASE_CORNER](#) enumeration;
- **OBJPROP_ANCHOR** - defines the anchor point in object itself and can be one of the 9 values of the [ENUM_ANCHOR_POINT](#) enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** - defines the object rotation angle counterclockwise.

The necessary variant can be specified using the function [ObjectSetInteger](#)(chart_handle, object_name, [OBJPROP_ANCHOR](#), anchor_point_mode), where anchor_point_mode is one of the values of [ENUM_ANCHOR_POINT](#).

ENUM_ANCHOR_POINT

ID	Description
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner

ANCHOR_LEFT	Anchor point to the left in the center
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner
ANCHOR_LOWER	Anchor point below in the center
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner
ANCHOR_RIGHT	Anchor point to the right in the center
ANCHOR_RIGHT_UPPER	Anchor point at the upper right corner
ANCHOR_UPPER	Anchor point above in the center
ANCHOR_CENTER	Anchor point strictly in the center of the object

The [OBJ_BUTTON](#), [OBJ_RECTANGLE_LABEL](#), [OBJ_EDIT](#) and [OBJ_CHART](#) objects have a fixed anchor point in the upper left corner (ANCHOR_LEFT_UPPER).

Example:

```
string text_name="my_OBJ_TEXT_object";
if(ObjectFind(0,text_name)<0)
{
    Print("Object ",text_name," not found. Error code = ",GetLastError());
    //--- Get the maximal price of the chart
    double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
    //--- Create object Label
    ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
    //--- Set color of the text
    ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
    //--- Set background color
    ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
    //--- Set text for the Label object
    ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent()));
    //--- Set text font
    ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
    //--- Set font size
    ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
    //--- Bind to the upper right corner
    ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
    //--- Rotate 90 degrees counter-clockwise
    ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
    //--- Forbid the selection of the object by mouse
    ObjectSetInteger(0,text_name,OBJPROP_SELECTABLE,false);
    //--- redraw object
    ChartRedraw(0);
}
```

Graphical objects Arrow (OBJ_ARROW) have only 2 ways of linking their coordinates. Identifiers are listed in ENUM_ARROW_ANCHOR.

ENUM_ARROW_ANCHOR

ID	Description
ANCHOR_TOP	Anchor on the top side
ANCHOR_BOTTOM	Anchor on the bottom side

Example:

```

void OnStart()
{
    //--- Auxiliary arrays
    double Ups[],Downs[];
    datetime Time[];
    //--- Set the arrays as timeseries
    ArraySetAsSeries(Ups,true);
    ArraySetAsSeries(Downs,true);
    ArraySetAsSeries(Time,true);
    //--- Create handle of the Indicator Fractals
    int FractalsHandle=iFractals(NULL,0);
    Print("FractalsHandle = ",FractalsHandle);
    //--- Set Last error value to Zero
    ResetLastError();
    //--- Try to copy the values of the indicator
    int copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
    if(copied<=0)
    {
        Print("Unable to copy the upper fractals. Error = ",GetLastError());
        return;
    }

    ResetLastError();
    //--- Try to copy the values of the indicator
    copied=CopyBuffer(FractalsHandle,1,0,1000,Downs);
    if(copied<=0)
    {
        Print("Unable to copy the bottom fractals. Error = ",GetLastError());
        return;
    }

    ResetLastError();
    //--- Copy timeseries containing the opening bars of the last 1000 ones
    copied=CopyTime(NULL,0,0,1000,Time);
    if(copied<=0)
    {
        Print("Unable to copy the Opening Time of the last 1000 bars");
        return;
    }

    int upcounter=0,downcounter=0; // count there the number of arrows
    bool created;// receive the result of attempts to create an object

```

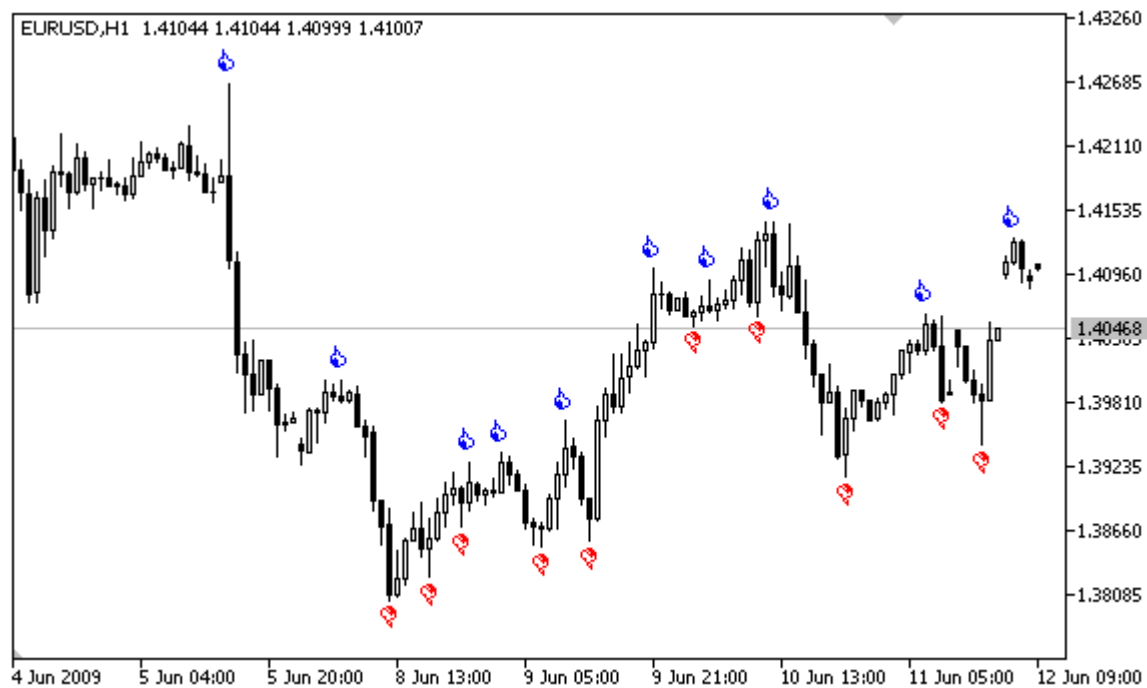


```

for(int i=2;i<copied;i++)// Run through the values of the indicator iFractals
{
    if(Ups[i]!=EMPTY_VALUE)// Found the upper fractal
    {
        if(upcounter<10)// Create no more than 10 "Up" arrows
        {
            //--- Try to create an "Up" object
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_UP,0,Time[i],Ups[i]);
            if(created)// If set up - let's make tuning for it
            {
                //--- Point anchor is below in order not to cover bar
                ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_BOTTOM);
                //--- Final touch - painted
                ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrBlue);
                upcounter++;
            }
        }
    }
    if(Downs[i]!=EMPTY_VALUE)// Found a lower fractal
    {
        if(downcounter<10)// Create no more than 10 arrows "Down"
        {
            //--- Try to create an object "Down"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_DOWN,0,Time[i],Downs[i]);
            if(created)// If set up - let's make tuning for it
            {
                //--- Point anchor is above in order not to cover bar
                ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_TOP);
                //--- Final touch - painted
                ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrRed);
                downcounter++;
            }
        }
    }
}
}

```

After the script execution the chart will look like in this figure.



The Chart Corner to Which an Object Is Attached

There is a number of [graphical objects](#) for which you can set a chart corner, relative to which the coordinates are specified in pixels. These are the following types of objects (in brackets object type identifiers are specified):

- Label (OBJ_LABEL);
- Button (OBJ_BUTTON);
- Bitmap Label (OBJ_BITMAP_LABEL);
- Edit (OBJ_EDIT).
- Rectangle Label (OBJ_RECTANGLE_LABEL);

Object	ID	X/Y	Width/ Height	Date/Price	OBJPROP_CORNER	OBJPROP_ANCHOR	OBJPROP_ANGLE
Text	OBJ_TEXT	—	—	Yes	—	Yes	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	—	Yes	Yes	Yes
Button	OBJ_BUTTON	Yes	Yes	—	Yes	—	—
Bitmap	OBJ_BITMAP	—	Yes (read only)	Yes	—	Yes	—
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	—	Yes	Yes	—
Edit	OBJ_EDIT	Yes	Yes	—	Yes	—	—
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	—	Yes	—	—

The following designations are used in the table:

- **X/Y** - coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** - objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** - anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** - defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the [ENUM_BASE_CORNER](#) enumeration;
- **OBJPROP_ANCHOR** - defines the anchor point in object itself and can be one of the 9 values of the [ENUM_ANCHOR_POINT](#) enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** - defines the object rotation angle counterclockwise.

In order to specify the chart corner, from which X and Y coordinates will be measured in pixels, use [ObjectSetInteger](#)(chartID, name, [OBJPROP_CORNER](#), chart_corner), where:

- chartID - chart identifier;
- name - name of a graphical object;
- OBJPROP_CORNER - property ID to specify the corner for binding;
- chart_corner - the desired chart corner, can be one of the values of the ENUM_BASE_CORNER enumeration.

ENUM_BASE_CORNER

ID	Description
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner of the chart

Example:

```

void CreateLabel(long   chart_id,
                string  name,
                int     chart_corner,
                int     anchor_point,
                string  text_label,
                int     x_ord,
                int     y_ord)
{
    //---
    if(ObjectCreate(chart_id,name,OBJ_LABEL,0,0,0))
    {
        ObjectSetInteger(chart_id,name,OBJPROP_CORNER,chart_corner);
        ObjectSetInteger(chart_id,name,OBJPROP_ANCHOR,anchor_point);
        ObjectSetInteger(chart_id,name,OBJPROP_XDISTANCE,x_ord);
        ObjectSetInteger(chart_id,name,OBJPROP_YDISTANCE,y_ord);
        ObjectSetString(chart_id,name,OBJPROP_TEXT,text_label);
    }
    else
        Print("Failed to create the object OBJ_LABEL ",name," Error code = ", GetLastError());
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //---
    int height=(int)ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);

```



```
int width=(int)ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
string arrows[4]={"LEFT_UPPER","RIGHT_UPPER","RIGHT_LOWER","LEFT_LOWER"};
CreateLabel(0,arrows[0],CORNER_LEFT_UPPER,ANCHOR_LEFT_UPPER,arrows[0],50,50);
CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,ANCHOR_RIGHT_UPPER,arrows[1],50,50);
CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,ANCHOR_RIGHT_LOWER,arrows[2],50,50);
CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,ANCHOR_LEFT_LOWER,arrows[3],50,50);
}
```


Visibility of Objects

The combination of object visibility flags determines chart timeframes, where the object is visible. To set/get the value of the OBJPROP_TIMEFRAMES property, you can use functions [ObjectSetInteger\(\)/ObjectGetInteger\(\)](#).

ID	Value	Description
OBJ_NO_PERIODS	0	The object is not drawn in all timeframes
OBJ_PERIOD_M1	0x00000001	The object is drawn in 1-minute chart
OBJ_PERIOD_M2	0x00000002	The object is drawn in 2-minute chart
OBJ_PERIOD_M3	0x00000004	The object is drawn in 3-minute chart
OBJ_PERIOD_M4	0x00000008	The object is drawn in 4-minute chart
OBJ_PERIOD_M5	0x00000010	The object is drawn in 5-minute chart
OBJ_PERIOD_M6	0x00000020	The object is drawn in 6-minute chart
OBJ_PERIOD_M10	0x00000040	The object is drawn in 10-minute chart
OBJ_PERIOD_M12	0x00000080	The object is drawn in 12-minute chart
OBJ_PERIOD_M15	0x00000100	The object is drawn in 15-minute chart
OBJ_PERIOD_M20	0x00000200	The object is drawn in 20-minute chart
OBJ_PERIOD_M30	0x00000400	The object is drawn in 30-minute chart
OBJ_PERIOD_H1	0x00000800	The object is drawn in 1-hour chart
OBJ_PERIOD_H2	0x00001000	The object is drawn in 2-hour chart
OBJ_PERIOD_H3	0x00002000	The object is drawn in 3-hour chart
OBJ_PERIOD_H4	0x00004000	The object is drawn in 4-hour chart

OBJ_PERIOD_H6	0x00008000	The object is drawn in 6-hour chart
OBJ_PERIOD_H8	0x00010000	The object is drawn in 8-hour chart
OBJ_PERIOD_H12	0x00020000	The object is drawn in 12-hour chart
OBJ_PERIOD_D1	0x00040000	The object is drawn in day charts
OBJ_PERIOD_W1	0x00080000	The object is drawn in week charts
OBJ_PERIOD_MN1	0x00100000	The object is drawn in month charts
OBJ_ALL_PERIODS	0x001fffff	The object is drawn in all timeframes

Visibility flags can be combined using the symbol "|", for example, the combination of flags OBJ_PERIOD_M10|OBJ_PERIOD_H4 means that the object will be visible on the 10-minute and 4-hour timeframes.

Example:

```
void OnStart()
{
//---
    string highlevel="PreviousDayHigh";
    string lowlevel="PreviousDayLow";
    double prevHigh;           // The previous day High
    double prevLow;            // The previous day Low
    double highs[],lows[];     // Arrays for High and Low

//--- Reset the last error
    ResetLastError();
//--- Get the last 2 High values on the daily timeframe
    int highsgot=CopyHigh(Symbol(),PERIOD_D1,0,2,highs);
    if(highsgot>0) // If copying was successful
    {
        Print("High prices for the last 2 days were obtained successfully");
        prevHigh=highs[0]; // The previous day High
        Print("prevHigh = ",prevHigh);
        if(ObjectFind(0,highlevel)<0) // Object with the name highlevel not found
        {
            ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // Create the Horizontal Line object
        }
        //--- Set value for the price level for the line highlevel
        ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
        //--- Set the visibility only PERIOD_M10 and PERIOD_H4
```



```

        ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
    }
    else
    {
        Print("Could not get High prices over the past 2 days, Error = ",GetLastError())
    }

//--- Reset the last error
ResetLastError();
//--- Get the 2 days values Low on the daily timeframe
int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2, lows);
if(lowsgot>0) // If copying was successful
{
    Print("Low prices for the last 2 days were obtained successfully");
    prevLow=lows[0]; // The previous day Low
    Print("prevLow = ",prevLow);
    if(ObjectFind(0,lowlevel)<0) // Object with the name lowlevel not found
    {
        ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // Create the Horizontal Line object
    }
    //--- Set value for the price level for the line lowlevel
    ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
    //--- Set the visibility only PERIOD_M10 and PERIOD_H4
    ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
}
else Print("Could not get Low prices for the last 2 days, Error = ",GetLastError())

ChartRedraw(0); // redraw the chart forcibly
}

```

See also

[PeriodSeconds](#), [Period](#), [Chart timeframes](#), [Date and Time](#)

Levels of Elliott Wave

Elliott Waves are represented by two graphical objects of types OBJ_ELLIOTWAVE5 and OBJ_ELLIOTWAVE3. To set the wave size (method of wave labeling), the OBJPROP_DEGREE property is used, to which one of values of the ENUM_ELLIOT_WAVE_DEGREE enumeration can be assigned.

ENUM_ELLIOT_WAVE_DEGREE

ID	Description
ELLIOTT_GRAND_SUPERCYCLE	Grand Supercycle
ELLIOTT_SUPERCYCLE	Supercycle
ELLIOTT_CYCLE	Cycle
ELLIOTT_PRIMARY	Primary
ELLIOTT_INTERMEDIATE	Intermediate
ELLIOTT_MINOR	Minor
ELLIOTT_MINUTE	Minute
ELLIOTT_MINUETTE	Minuette
ELLIOTT_SUBMINUETTE	Subminuette

Example:

```
for(int i=0;i<ObjectsTotal(0);i++)
{
    string currobj=ObjectName(0,i);
    if((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE3) ||
        ((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE5)))
    {
        //--- set the marking level in INTERMEDIATE
        ObjectSetInteger(0,currobj,OBJPROP_DEGREE,ELLIOTT_INTERMEDIATE);
        //--- show lines between tops of waves
        ObjectSetInteger(0,currobj,OBJPROP_DRAWLINES,true);
        //--- set line color
        ObjectSetInteger(0,currobj,OBJPROP_COLOR,clrBlue);
        //--- set line width
        ObjectSetInteger(0,currobj,OBJPROP_WIDTH,5);
        //--- set description
        ObjectSetString(0,currobj,OBJPROP_TEXT,"test script");
    }
}
```


Gann Objects

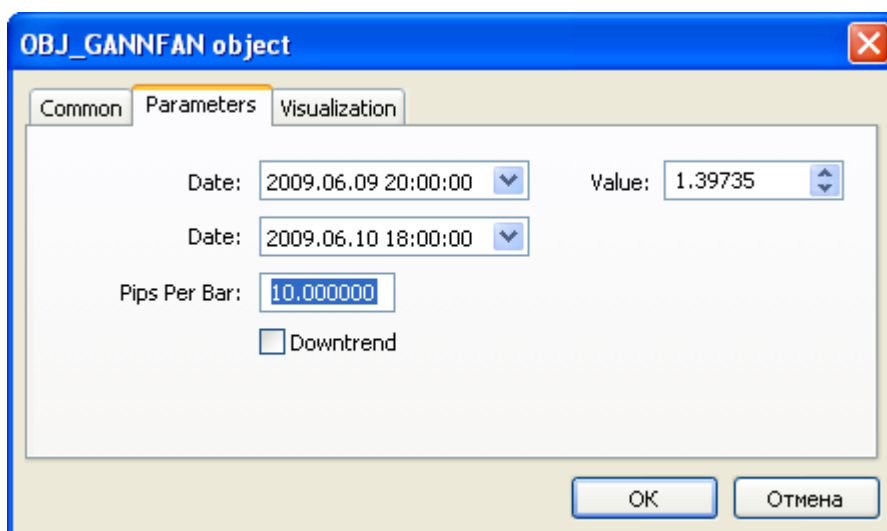
For Gann Fan (OBJ_GANNFAN) and Gann Grid (OBJ_GANNGRID) objects you can specify two values of the ENUM_GANN_DIRECTION enumeration that sets the trend direction.

ENUM_GANN_DIRECTION

ID	Description
GANN_UP_TREND	Line corresponding to the uptrend line
GANN_DOWN_TREND	Line corresponding to the downward trend

To set the scale of the main line as 1x1, use function [ObjectSetDouble](#)(chart_handle, gann_object_name, OBJPROP_SCALE, scale), where:

- chart_handle - chart window where the object is located;
- gann_object_name - object name;
- OBJPROP_SCALE - identifier of the "Scale" property;
- scale - required scale in units of Pips/Bar.



Example of creating Gann Fan:

```
void OnStart()
{
    //---
    string my_gann="OBJ_GANNFAN object";
    if(ObjectFind(0,my_gann)<0) // Object not found
    {
        //--- Inform about the failure
        Print("Object ",my_gann," not found. Error code = ",GetLastError());
        //--- Get the maximal price of the chart
        double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
        //--- Get the minimal price of the chart
        double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
        //--- How many bars are shown in the chart?
```



```

int bars_on_chart=ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- Create an array, to write the opening time of each bar to
datetime Time[];
//--- Arrange access to the array as that of timeseries
ArraySetAsSeries(Time,true);
//--- Now copy data of bars visible in the chart into this array
int times=CopyTime(NULL,0,0,bars_on_chart,Time);
if(times<=0)
{
    Print("Could not copy the array with the open time!");
    return;
}
//--- Preliminary preparations completed

//--- Index of the central bar in the chart
int center_bar=bars_on_chart/2;
//--- Chart equator - between the maximum and minimum
double mean=(chart_max_price+chart_min_price)/2.0;
//--- Set the coordinates of the first anchor point to the center
ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Time[center_bar],mean,
    //--- Second anchor point to the right
    Time[center_bar/2],(mean+chart_min_price)/2.0);
Print("Time[center_bar] = "+(string)Time[center_bar]+"   Time[center_bar/2] = "+
//Print("Time[center_bar]/="+Time[center_bar]+"   Time[center_bar/2]="+Time[cente
//--- Set the scale in units of Pips / Bar
ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
//--- Set the line trend
ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
//--- Set the line width
ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
//--- Define the line style
ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
//--- Set the line color
ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
//--- Allow the user to select an object
ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
//--- Select it yourself
ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
//--- Draw it on the chart
ChartRedraw(0);
}
}

```


Web Colors

The following color constants are defined for the [color](#) type:

						clrDarkTurquoise	
clrLightSeaGreen							
clrGoldenrod	clrMediumSpringGreen	clrLawnGreen					
	clrOrange	clrGold	clrYellow	clrChartreuse	clrLime	clrSpringGreen	clrAqua
clrDeepSkyBlue		clrMagenta					
		clrMediumTurquoise		clrTurquoise			clrDarkKhaki
		clrGreenYellow	clrMediumAquamarine				clrOrchid
clrMediumPurple				clrDarkGray	clrSandyBrown		clrTan
	clrBurlyWood	clrHotPink		clrViolet		clrSkyBlue	clrLightSalmon
clrPlum	clrKhaki	clrLightGreen	clrAquamarine	clrSilver	clrLightSkyBlue	clrLightSteelBlue	clrLightBlue
clrPaleGreen	clrThistle	clrPowderBlue	clrPaleGoldenrod	clrPaleTurquoise	clrLightGray	clrWheat	clrNavajoWhite
clrMoccasin	clrLightPink	clrGainsboro	clrPeachPuff	clrPink	clrBisque	clrLightGoldenrod	clrBlanchedAlmond
clrLemonChiffon	clrBeige	clrAntiqueWhite	clrPapayaWhip	clrCornsilk	clrLightYellow	clrLightCyan	clrLinen
clrLavender	clrMistyRose	clrOldLace	clrWhiteSmoke	clrSeashell	clrIvory	clrHoneydew	clrAliceBlue
clrLavenderBlush	clrMintCream	clrSnow	clrWhite				

Color can be set to an object using the [ObjectSetInteger\(\)](#) function. For setting color to custom indicators the [PlotIndexSetInteger\(\)](#) function is used. For getting color values there are similar functions [ObjectGetInteger\(\)](#) and [PlotIndexGetInteger\(\)](#).

Example:

```
//---- indicator settings
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 3
#property indicator_type1 DRAW_LINE
#property indicator_type2 DRAW_LINE
#property indicator_type3 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_color2 clrRed
#property indicator_color3 clrLime
```


Wingdings

Characters of Wingdings used with the [OBJ_ARROW](#) object:

32		33		34		35		36		37		38		39		40		41		42		43		44		45		46		47	
48		49		50		51		52		53		54		55		56		57		58		59		60		61		62		63	
64		65		66		67		68		69		70		71		72		73		74		75		76		77		78		79	
80		81		82		83		84		85		86		87		88		89		90		91		92		93		94		95	
96		97		98		99		100		101		102		103		104		105		106		107		108		109		110		111	
112		113		114		115		116		117		118		119		120		121		122		123		124		125		126		127	
128		129		130		131		132		133		134		135		136		137		138		139		140		141		142		143	
144		145		146		147		148		149		150		151		152		153		154		155		156		157		158		159	
160		161		162		163		164		165		166		167		168		169		170		171		172		173		174		175	
176		177		178		179		180		181		182		183		184		185		186		187		188		189		190		191	
192		193		194		195		196		197		198		199		200		201		202		203		204		205		206		207	
208		209		210		211		212		213		214		215		216		217		218		219		220		221		222		223	
224		225		226		227		228		229		230		231		232		233		234		235		236		237		238		239	
240		241		242		243		244		245		246		247		248		249		250		251		252		253		254		255	

A necessary character can be set using the [ObjectSetInteger\(\)](#) function.

Example:

```
void OnStart()
{
    //---
    string up_arrow="up_arrow";
    datetime time=TimeCurrent();
    double lastClose[1];
    int close=CopyClose(Symbol(),Period(),0,1,lastClose);    // Get the Close price
    //--- If the price was obtained
    if(close>0)
    {
        ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0,0);    // Create an arrow
        ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);    // Set the arrow code
        ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);    // Set time
        ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);    // Set price
        ChartRedraw(0);    // Draw arrow now
    }
    else
        Print("Unable to get the latest Close price!");
}
```


Indicators Constants

There are 37 predefined [technical indicators](#), which can be used in programs written in the MQL5 language. In addition, there is an opportunity to create custom indicators using the [iCustom\(\)](#) function. All constants required for that are divided into 5 groups:

- [Price constants](#) - for selecting the type of price or volume, on which an indicator is calculated;
- [Smoothing methods](#) - built-in smoothing methods used in indicators;
- [Indicator lines](#) - identifiers of indicator buffers when accessing indicator values using [CopyBuffer\(\)](#);
- [Drawing styles](#) - for indicating one of 18 types of drawing and setting the line drawing style;
- [Custom indicators properties](#) are used in functions for working with [custom](#) indicators;
- [Types of indicators](#) are used for specifying the type of technical indicator when creating a handle using [IndicatorCreate\(\)](#);
- [Identifiers of data types](#) are used for specifying the type of data passed in an array of the [MqlParam](#) type into the [IndicatorCreate\(\)](#) function.

Price Constants

Calculations of technical indicators require price values and/or values of volumes, on which calculations will be performed. There are 7 predefined identifiers from the ENUM_APPLIED_PRICE enumeration, used to specify the desired price base for calculations.

ENUM_APPLIED_PRICE

ID	Description
PRICE_CLOSE	Close price
PRICE_OPEN	Open price
PRICE_HIGH	The maximum price for the period
PRICE_LOW	The minimum price for the period
PRICE_MEDIAN	Median price, (high + low)/2
PRICE_TYPICAL	Typical price, (high + low + close)/3
PRICE_WEIGHTED	Average price, (high + low + close + close)/4

If the volume is used in calculations, it's necessary to specify one of the two values from the ENUM_APPLIED_VOLUME enumeration.

ENUM_APPLIED_VOLUME

ID	Description
VOLUME_TICK	Tick volume
VOLUME_REAL	Trade volume

The [iStochastic\(\)](#) technical Indicator can be calculated in two ways using:

- either only Close prices;
- or High and Low prices.

To select a necessary variant for calculation, specify one of the values of the ENUM_STO_PRICE enumeration.

ENUM_STO_PRICE

ID	Description
STO_LOWHIGH	Calculation is based on Low/High prices
STO_CLOSECLOSE	Calculation is based on Close/Close prices

If a technical indicator uses for calculations price data, type of which is set by ENUM_APPLIED_PRICE, then handle of any indicator (built in the terminal or written by a user) can be used as the input price series. In this case, values of the zero buffer of the indicator will be used for calculations. This makes it easy to build values of one indicator using values of another indicator. The handle of a custom indicator is created by calling the [iCustom\(\)](#) function.

Example:

```

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- input parameters
input int      RSIPeriod=14;          // Period for calculating the RSI
input int      Smooth=8;              // Smoothing period RSI
input ENUM_MA_METHOD meth=MODE_SMA;  // Method of smoothing
//---- plot RSI
#property indicator_label1  "RSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//---- plot RSI_Smoothed
#property indicator_label2  "RSI_Smoothed"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrNavy
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- indicator buffers
double        RSIBuffer[];           // Here we store the values of RSI
double        RSI_SmoothedBuffer[]; // Here will be smoothed values of RSI
int           RSIhandle;              // Handle to the RSI indicator
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,RSIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,RSI_SmoothedBuffer,INDICATOR_DATA);
    IndicatorSetString(INDICATOR_SHORTNAME,"iRSI");
    IndicatorSetInteger(INDICATOR_DIGITS,2);
//---
    RSIhandle=iRSI(NULL,0,RSIPeriod,PRICE_CLOSE);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const int begin,
               const double &price[]
               )
{

```



```
//--- Reset the value of the last error
ResetLastError();
//--- Get RSI indicator data in an array RSIBuffer []
int copied=CopyBuffer(RSIhandle,0,0,rates_total,RSIBuffer);
if(copied<=0)
{
    Print("Unable to copy the values of the indicator RSI. Error = ",
        GetLastError()," copied =",copied);
    return(0);
}
//--- Create the indicator of average values using values of RSI
int RSI_MA_handle=iMA(NULL,0,Smooth,0,method,RSIhandle);
copied=CopyBuffer(RSI_MA_handle,0,0,rates_total,RSI_SmoothedBuffer);
if(copied<=0)
{
    Print("Unable to copy the smoothed indicator of RSI. Error = ",
        GetLastError()," copied =",copied);
    return(0);
}
//--- return value of prev_calculated for next call
return(rates_total);
}
```


Smoothing Methods

Many technical indicators are based on various methods of the price series smoothing. Some standard technical indicators require specification of the smoothing type as an input parameter. For specifying the desired type of smoothing, identifiers listed in the ENUM_MA_METHOD enumeration are used.

ENUM_MA_METHOD

ID	Description
MODE_SMA	Simple averaging
MODE_EMA	Exponential averaging
MODE_SMMA	Smoothed averaging
MODE_LWMA	Linear-weighted averaging

Example:

```
double ExtJaws[];
double ExtTeeth[];
double ExtLips[];
//---- handles for moving averages
int    ExtJawsHandle;
int    ExtTeethHandle;
int    ExtLipsHandle;
//--- get MA's handles
ExtJawsHandle=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtTeethHandle=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtLipsHandle=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
```


Indicators Lines

Some [technical indicators](#) have several buffers drawn in the chart. Numbering of indicator buffers starts with 0. When copying indicator values using the [CopyBuffer\(\)](#) function into an array of the double type, for some indicators one may indicate the identifier of a copied buffer instead of its number.

Identifiers of indicator lines permissible when copying values of [iMACD\(\)](#), [iRVI\(\)](#) and [iStochastic\(\)](#).

Constant	Value	Description
MAIN_LINE	0	Main line
SIGNAL_LINE	1	Signal line

Identifiers of indicator lines permissible when copying values of [ADX\(\)](#) and [ADXW\(\)](#).

Constant	Value	Description
MAIN_LINE	0	Main line
PLUSDI_LINE	1	Line +DI
MINUSDI_LINE	2	Line -DI

Identifiers of indicator lines permissible when copying values of [iBands\(\)](#).

Constant	Value	Description
BASE_LINE	0	Main line
UPPER_BAND	1	Upper limit
LOWER_BAND	2	Lower limit

Identifiers of indicator lines permissible when copying values of [iEnvelopes\(\)](#) and [iFractals\(\)](#).

Constant	Value	Description
UPPER_LINE	0	Upper line
LOWER_LINE	1	Bottom line

Identifiers of indicator lines permissible when copying values of [iGator\(\)](#)

Constant	Value	Description
UPPER_HISTOGRAM	0	Upper histogram
LOWER_HISTOGRAM	2	Bottom histogram

Identifiers of indicator lines permissible when copying values of [iAlligator\(\)](#).

Constant	Value	Description
----------	-------	-------------

GATORJAW_LINE	0	Jaw line
GATORTEETH_LINE	1	Teeth line
GATORLIPS_LINE	2	Lips line

Identifiers of indicator lines permissible when copying values of [ilchimoku\(\)](#).

Constant	Value	Description
TENKANSEN_LINE	0	Tenkan-sen line
KIJUNSEN_LINE	1	Kijun-sen line
SENKOSPAN_A_LINE	2	Senkou Span A line
SENKOSPAN_B_LINE	3	Senkou Span B line
CHIKOSPAN_LINE	4	Chikou Span line

Drawing Styles

When creating [a custom indicator](#), you can specify one of 18 types of graphical plotting (as displayed in the main chart window or a chart subwindow), whose values are specified in the ENUM_DRAW_TYPE enumeration.

In one custom indicator, it is permissible to use any [indicator building/drawing types](#). Each construction type requires specification of one to five [global arrays](#) for storing data necessary for drawing. These data arrays must be bound with indicator buffers using the [SetIndexBuffer\(\)](#) function. The type of data from [ENUM_INDEXBUFFER_TYPE](#) should be specified for each buffer.

Depending on the drawing style, you may need one to four value buffers (marked as INDICATOR_DATA). If a style admits dynamic alternation of colors (all styles contain COLOR in their names), then you'll need one more buffer of color (indicated type INDICATOR_COLOR_INDEX). The color buffers are always bound after value buffers corresponding to the style.

ENUM_DRAW_TYPE

ID	Description	Data buffers	Color buffers
DRAW_NONE	Not drawn	1	0
DRAW_LINE	Line	1	0
DRAW_SECTION	Section	1	0
DRAW_HISTOGRAM	Histogram from the zero line	1	0
DRAW_HISTOGRAM2	Histogram of the two indicator buffers	2	0
DRAW_ARROW	Drawing arrows	1	0
DRAW_ZIGZAG	Style Zigzag allows vertical section on the bar	2	0
DRAW_FILLING	Color fill between the two levels	2	0
DRAW_BARS	Display as a sequence of bars	4	0
DRAW_CANDLES	Display as a sequence of candlesticks	4	0
DRAW_COLOR_LINE	Multicolored line	1	1
DRAW_COLOR_SECTION	Multicolored section	1	1
DRAW_COLOR_HISTOGRAM	Multicolored histogram from the zero line	1	1
DRAW_COLOR_HISTOGRAM2	Multicolored histogram of the two indicator	2	1

	buffers		
<u>DRAW_COLOR_ARROW</u>	Drawing multicolored arrows	1	1
<u>DRAW_COLOR_ZIGZAG</u>	Multicolored ZigZag	2	1
<u>DRAW_COLOR_BARS</u>	Multicolored bars	4	1
<u>DRAW_COLOR_CANDLES</u>	Multicolored candlesticks	4	1

To refine the display of the selected drawing type identifiers listed in ENUM_PLOT_PROPERTY are used.

For functions [PlotIndexSetInteger\(\)](#) and [PlotIndexGetInteger\(\)](#)

ENUM_PLOT_PROPERTY_INTEGER

ID	Description	Property type
PLOT_ARROW	Arrow code for style DRAW_ARROW	uchar
PLOT_ARROW_SHIFT	Vertical shift of arrows for style DRAW_ARROW	int
PLOT_DRAW_BEGIN	Number of initial bars without drawing and values in the DataWindow	int
PLOT_DRAW_TYPE	Type of graphical construction	<u>ENUM_DRAW_TYPE</u>
PLOT_SHOW_DATA	Sign of display of construction values in the DataWindow	bool
PLOT_SHIFT	Shift of indicator plotting along the time axis in bars	int
PLOT_LINE_STYLE	Drawing line style	<u>ENUM_LINE_STYLE</u>
PLOT_LINE_WIDTH	The thickness of the drawing line	int
PLOT_COLOR_INDEXES	The number of colors	int
PLOT_LINE_COLOR	The index of a buffer containing the drawing color	color modifier = index number of colors

For the function [PlotIndexSetDouble\(\)](#)

ENUM_PLOT_PROPERTY_DOUBLE

ID	Description	Property type
----	-------------	---------------

PLOT_EMPTY_VALUE	An empty value for plotting, for which there is no drawing	double
------------------	--	--------

For the function [PlotIndexSetString\(\)](#)

ENUM_PLOT_PROPERTY_STRING

ID	Description	Property type
PLOT_LABEL	The name of the indicator graphical series to display in the DataWindow. When working with complex graphical styles requiring several indicator buffers for display, the names for each buffer can be specified using ";" as a separator. Sample code is shown in DRAW_CANDLES	string

5 styles can be used for drawing lines in custom indicators. They are valid only for the line thickness 0 or 1.

ENUM_LINE_STYLE

ID	Description
STYLE_SOLID	Solid line
STYLE_DASH	Broken line
STYLE_DOT	Dotted line
STYLE_DASHDOT	Dash-dot line
STYLE_DASHDOTDOT	Dash - two points

To set the line drawing style and the type of drawing, the [PlotIndexSetInteger\(\)](#) function is used. For the Fibonacci extensions the thickness and drawing style of levels can be indicated using the [ObjectSetInteger\(\)](#) function.

Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- indicator buffers
double      MABuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
```



```

//--- Bind the Array to the indicator buffer with index 0
    SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
//--- Set the line drawing
    PlotIndexSetInteger(0,PLOT_DRAW_TYPE,DRAW_LINE);
//--- Set the style line
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,STYLE_DOT);
//--- Set line color
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrRed);
//--- Set line thickness
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
//--- Set labels for the line
    PlotIndexSetString(0,PLOT_LABEL,"Moving Average");
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//---
    for(int i=prev_calculated;i<rates_total;i++)
    {
        MABuffer[i]=close[i];
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```


Custom Indicators Properties

The number of indicator buffers that can be used in a custom indicator is unlimited. But for each array, which is designated as the indicator buffer using the [SetIndexBuffer\(\)](#) function, it's necessary to specify the data type that it will store. This may be one of the values of the ENUM_INDEXBUFFER_TYPE enumeration.

ENUM_INDEXBUFFER_TYPE

ID	Description
INDICATOR_DATA	Data to draw
INDICATOR_COLOR_INDEX	Color
INDICATOR_CALCULATIONS	Auxiliary buffers for intermediate calculations

A custom indicator has a lot of settings to provide convenient displaying. These settings are made through the assignment of corresponding indicator properties using functions [IndicatorSetDouble\(\)](#), [IndicatorSetInteger\(\)](#) and [IndicatorSetString\(\)](#). Identifiers of indicator properties are listed in the ENUM_CUSTOMIND_PROPERTY enumeration.

ENUM_CUSTOMIND_PROPERTY_INTEGER

ID	Description	Property type
INDICATOR_DIGITS	Accuracy of drawing of indicator values	int
INDICATOR_HEIGHT	Fixed height of the indicator's window (the preprocessor command #property indicator_height)	int
INDICATOR_LEVELS	Number of levels in the indicator window	int
INDICATOR_LEVELCOLOR	Color of the level line	color modifier = level number
INDICATOR_LEVELSTYLE	Style of the level line	ENUM_LINE_STYLE modifier = level number
INDICATOR_LEVELWIDTH	Thickness of the level line	int modifier = level number

ENUM_CUSTOMIND_PROPERTY_DOUBLE

ID	Description	Property type
INDICATOR_MINIMUM	Minimum of the indicator window	double
INDICATOR_MAXIMUM	Maximum of the indicator window	double

INDICATOR_LEVELVALUE	Level value	double level number	modifier =
----------------------	-------------	------------------------	------------

ENUM_CUSTOMIND_PROPERTY_STRING

ID	Description	Property type	
INDICATOR_SHORTNAME	Short indicator name	string	
INDICATOR_LEVELTEXT	Level description	string level number	modifier =

Examples:

```
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 2
#property indicator_type1 DRAW_LINE
#property indicator_type2 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_color2 clrRed
//--- input parameters
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//--- indicator buffers
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,MainBuffer,INDICATOR_DATA);
SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
SetIndexBuffer(2,HighesBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(3,LowesBuffer,INDICATOR_CALCULATIONS);
//--- set accuracy
IndicatorSetInteger(INDICATOR_DIGITS,2);
//--- set levels
IndicatorSetInteger(INDICATOR_LEVELS,2);
IndicatorSetDouble(INDICATOR_LEVELVALUE,0,20);
IndicatorSetDouble(INDICATOR_LEVELVALUE,1,80);
//--- set maximum and minimum for subwindow
IndicatorSetDouble(INDICATOR_MINIMUM,0);
IndicatorSetDouble(INDICATOR_MAXIMUM,100);
```



```
//--- sets first bar from which index will be drawn
PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,KPeriod+Slowing-2);
PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,KPeriod+Slowing+DPeriod);
//--- set style STYLE_DOT for second line
PlotIndexSetInteger(1,PLOT_LINE_STYLE,STYLE_DOT);
//--- name for DataWindow and indicator subwindow label
IndicatorSetString(INDICATOR_SHORTNAME,"Stoch("+KPeriod+", "+DPeriod+", "+Slowing+")");
PlotIndexSetString(0,PLOT_LABEL,"Main");
PlotIndexSetString(1,PLOT_LABEL,"Signal");
//--- sets drawing line to empty value
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0.0);
//--- initialization done
}
```


Types of Technical Indicators

There are two ways to create an indicator handle for further [accessing to its values](#). The first way is to directly specify a function name from the list of [technical indicators](#). The second method using the [IndicatorCreate\(\)](#) is to uniformly create a handle of any indicator by assigning an identifier from the ENUM_INDICATOR enumeration. Both ways of handle creation are equal, you can use the one that is most convenient in a particular case when writing a program in MQL5.

When creating an indicator of type IND_CUSTOM, the *type* field of the first element of an array of [input parameters MqlParam](#) must have the TYPE_STRING value of the enumeration [ENUM_DATATYPE](#), while the field *string_value* of the first element must contain the name of the custom indicator.

ENUM_INDICATOR

Identifier	Indicator
IND_AC	Accelerator Oscillator
IND_AD	Accumulation/Distribution
IND_ADX	Average Directional Index
IND_ADXW	ADX by Welles Wilder
IND_ALLIGATOR	Alligator
IND_AMA	Adaptive Moving Average
IND_AO	Awesome Oscillator
IND_ATR	Average True Range
IND_BANDS	Bollinger Bands®
IND_BEARS	Bears Power
IND_BULLS	Bulls Power
IND_BWMFI	Market Facilitation Index
IND_CCI	Commodity Channel Index
IND_CHAIKIN	Chaikin Oscillator
IND_CUSTOM	Custom indicator
IND_DEMA	Double Exponential Moving Average
IND_DEMARKER	DeMarker
IND_ENVELOPES	Envelopes
IND_FORCE	Force Index
IND_FRACTALS	Fractals
IND_FRAMA	Fractal Adaptive Moving Average
IND_GATOR	Gator Oscillator

IND_ICHIMOKU	Ichimoku Kinko Hyo
IND_MA	Moving Average
IND_MACD	MACD
IND_MFI	Money Flow Index
IND_MOMENTUM	Momentum
IND_OBV	On Balance Volume
IND_OSMA	OsMA
IND_RSI	Relative Strength Index
IND_RVI	Relative Vigor Index
IND_SAR	Parabolic SAR
IND_STDDEV	Standard Deviation
IND_STOCHASTIC	Stochastic Oscillator
IND_TEMA	Triple Exponential Moving Average
IND_TRIX	Triple Exponential Moving Averages Oscillator
IND_VIDYA	Variable Index Dynamic Average
IND_VOLUMES	Volumes
IND_WPR	Williams' Percent Range

Data Type Identifiers

When creating an indicator handle using the [IndicatorCreate\(\)](#) function, an array of [MqlParam](#) type must be specified as the last parameter. Accordingly, the [MqlParam](#) structure, describing indicator, contains a special field *type*. This field contains information about the data type ([real](#), [integer](#) or [string](#) type) that are passed by a particular element of the array. The value of this field of the [MqlParam](#) structure may be one of [ENUM_DATATYPE](#) values.

ENUM_DATATYPE

Identifier	Data type
TYPE_BOOL	bool
TYPE_CHAR	char
TYPE_UCHAR	uchar
TYPE_SHORT	short
TYPE_USHORT	ushort
TYPE_COLOR	color
TYPE_INT	int
TYPE_UINT	uint
TYPE_DATETIME	datetime
TYPE_LONG	long
TYPE_ULONG	ulong
TYPE_FLOAT	float
TYPE_DOUBLE	double
TYPE_STRING	string

Each element of the array describes the corresponding input parameter of a created [technical indicator](#), so the type and order of elements in the array must be strictly maintained in accordance with the description.

Environment State

Constants describing the current runtime environment of an mql5-program are divided into groups:

- [Client terminal properties](#) - information about the client terminal;
- [Executed MQL5-program properties](#) - mql5 program properties, which help to control its execution;
- [Symbol properties](#) - obtaining information about a symbol;
- [Account properties](#) - information about the current account;
- [Testing Statistics](#) - results of Expert Advisor testing.

Client Terminal Properties

Information about the client terminal can be obtained by two functions: [TerminalInfoInteger\(\)](#) and [TerminalInfoString\(\)](#). For parameters, these functions accept values from ENUM_TERMINAL_INFO_INTEGER and ENUM_TERMINAL_INFO_STRING respectively.

ENUM_TERMINAL_INFO_INTEGER

Identifier	Description	Type
TERMINAL_BUILD	The client terminal build number	int
TERMINAL_COMMUNITY_ACCOUNT	The flag indicates the presence of MQL5.community authorization data in the terminal	bool
TERMINAL_COMMUNITY_CONNECTION	Connection to MQL5.community	bool
TERMINAL_CONNECTED	Connection to a trade server	bool
TERMINAL_DLLS_ALLOWED	Permission to use DLL	bool
TERMINAL_TRADE_ALLOWED	Permission to trade	bool
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the terminal settings	bool
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	bool
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to smartphone	bool
TERMINAL_MAXBARS	The maximal bars count on the chart	int
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID data for Push notifications	bool
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	int
TERMINAL_CPU_CORES	The number of CPU cores in the system	int
TERMINAL_DISK_SPACE	Free disk space for the MQL5\Files folder of the	int

	terminal (agent), MB	
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, MB	int
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal (agent), MB	int
TERMINAL_MEMORY_AVAILABLE	Free memory of the terminal (agent) process, MB	int
TERMINAL_MEMORY_USED	Memory used by the terminal (agent), MB	int
TERMINAL_X64	Indication of the "64-bit terminal"	bool
TERMINAL_OPENCL_SUPPORT	The version of the supported OpenCL in the format of 0x00010002 = 1.2. "0" means that OpenCL is not supported	int
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI). Knowing the parameter value, you can set the size of graphical objects so that they look the same on monitors with different resolution characteristics.	int
TERMINAL_PING_LAST	The last known value of a ping to a trade server in microseconds. One second comprises of one million microseconds	int
Key identifier	Description	
TERMINAL_KEYSTATE_LEFT	State of the "Left arrow" key	int
TERMINAL_KEYSTATE_UP	State of the "Up arrow" key	int
TERMINAL_KEYSTATE_RIGHT	State of the "Right arrow" key	int
TERMINAL_KEYSTATE_DOWN	State of the "Down arrow" key	int
TERMINAL_KEYSTATE_SHIFT	State of the "Shift" key	int
TERMINAL_KEYSTATE_CONTROL	State of the "Ctrl" key	int
TERMINAL_KEYSTATE_MENU	State of the "Windows" key	int

TERMINAL_KEYSTATE_CAPSLOCK	State of the "CapsLock" key	int
TERMINAL_KEYSTATE_NUMLOCK	State of the "NumLock" key	int
TERMINAL_KEYSTATE_SCROLLLOCK	State of the "ScrollLock" key	int
TERMINAL_KEYSTATE_ENTER	State of the "Enter" key	int
TERMINAL_KEYSTATE_INSERT	State of the "Insert" key	int
TERMINAL_KEYSTATE_DELETE	State of the "Delete" key	int
TERMINAL_KEYSTATE_HOME	State of the "Home" key	int
TERMINAL_KEYSTATE_END	State of the "End" key	int
TERMINAL_KEYSTATE_TAB	State of the "Tab" key	int
TERMINAL_KEYSTATE_PAGEUP	State of the "PageUp" key	int
TERMINAL_KEYSTATE_PAGEDOWN	State of the "PageDown" key	int
TERMINAL_KEYSTATE_ESCAPE	State of the "Escape" key	int

Call to `TerminalInfoInteger(TERMINAL_KEYSTATE_XXX)` returns the same state code of a key as the [GetKeyState\(\)](#) function in MSDN.

Example of scaling factor calculation:

```
//--- Creating a 1.5 inch wide button on a screen
int screen_dpi = TerminalInfoInteger(TERMINAL_SCREEN_DPI); // Find DPI of the user monitor
int base_width = 144; // The basic width in the standard settings
int width = (button_width * screen_dpi) / 96; // Calculate the button width
...

//--- Calculating the scaling factor as a percentage
int scale_factor = (TerminalInfoInteger(TERMINAL_SCREEN_DPI) * 100) / 96;
//--- Use of the scaling factor
width = (base_width * scale_factor) / 100;
```

In the above example, the graphical [resource](#) looks the same on monitors with different resolution characteristics. The size of control elements (buttons, dialog windows, etc.) corresponds to personalization settings.

ENUM_TERMINAL_INFO_DOUBLE

Identifier	Description	Type
------------	-------------	------

TERMINAL_COMMUNITY_BALANCE	Balance in MQL5.community	double
----------------------------	---------------------------	--------

[File operations](#) can be performed only in two directories; corresponding paths can be obtained using the request for TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH properties.

ENUM_TERMINAL_INFO_STRING

Identifier	Description	Type
TERMINAL_LANGUAGE	Language of the terminal	string
TERMINAL_COMPANY	Company name	string
TERMINAL_NAME	Terminal name	string
TERMINAL_PATH	Folder from which the terminal is started	string
TERMINAL_DATA_PATH	Folder in which terminal data are stored	string
TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on a computer	string

For a better understanding of paths, stored in properties of TERMINAL_PATH, TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH parameters, it is recommended to execute the script, which will return these values for the current copy of the client terminal, installed on your computer

Example: Script returns information about the client terminal paths

```
//+-----+
//|                                     Check_TerminalPaths.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //---
    Print("TERMINAL_PATH = ", TerminalInfoString(TERMINAL_PATH));
    Print("TERMINAL_DATA_PATH = ", TerminalInfoString(TERMINAL_DATA_PATH));
    Print("TERMINAL_COMMONDATA_PATH = ", TerminalInfoString(TERMINAL_COMMONDATA_PATH));
}
```

As result of the script execution in the Experts Journal you will see a messages, like the following:

Message

TERMINAL_COMMONDATA_PATH = C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal...

TERMINAL_DATA_PATH = C:\Program Files\MetaTrader 5

TERMINAL_PATH = C:\Program Files\MetaTrader 5

Running MQL5 Program Properties

To obtain information about the currently running mql5 program, constants from ENUM_MQL_INFO_INTEGER and ENUM_MQL_INFO_STRING are used.

For function [MQLInfoInteger](#)

ENUM_MQL_INFO_INTEGER

Identifier	Description	Type
MQL_MEMORY_LIMIT	Maximum possible amount of dynamic memory for MQL5 program in MB	int
MQL_MEMORY_USED	The memory size used by MQL5 program in MB	int
MQL_PROGRAM_TYPE	Type of the mql5 program	ENUM_PROGRAM_TYPE
MQL_DLLS_ALLOWED	The permission to use DLL for the given executed program	bool
MQL_TRADE_ALLOWED	The permission to trade for the given executed program	bool
MQL_SIGNALS_ALLOWED	The permission to modify the Signals for the given executed program	bool
MQL_DEBUG	The flag, that indicates the debug mode	bool
MQL_PROFILER	The flag, that indicates the program operating in the code profiling mode	bool
MQL_TESTER	The flag, that indicates the tester process	bool
MQL_OPTIMIZATION	The flag, that indicates the optimization process	bool
MQL_VISUAL_MODE	The flag, that indicates the visual tester process	bool
MQL_FRAME_MODE	The flag, that indicates the Expert Advisor operating in gathering optimization result frames mode	bool
MQL_LICENSE_TYPE	Type of license of the EX5 module. The license refers to the EX5 module, from which a request is made using	ENUM_LICENSE_TYPE

	MQLInfoInteger(MQL_LICENSE_TYPE).	
--	-----------------------------------	--

For function [MQLInfoString](#)

ENUM_MQL_INFO_STRING

Identifier	Description	Type
MQL_PROGRAM_NAME	Name of the mql5-program executed	string
MQL5_PROGRAM_PATH	Path for the given executed program	string

For information about the type of the running program, values of ENUM_PROGRAM_TYPE are used.

ENUM_PROGRAM_TYPE

Identifier	Description
PROGRAM_SCRIPT	Script
PROGRAM_EXPERT	Expert
PROGRAM_INDICATOR	Indicator

ENUM_LICENSE_TYPE

Identifier	Description
LICENSE_FREE	A free unlimited version
LICENSE_DEMO	A trial version of a paid product from the Market. It works only in the strategy tester
LICENSE_FULL	A purchased licensed version allows at least 5 activations. The number of activations is specified by seller. Seller may increase the allowed number of activations
LICENSE_TIME	A version with a limited term license

Example:

```
ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PROGRAM_TYPE);
switch(mql_program)
{
    case PROGRAM_SCRIPT:
    {
        Print(__FILE__+" is script");
        break;
    }
}
```



```
    }  
    case PROGRAM_EXPERT:  
    {  
        Print(__FILE__+" is Expert Advisor");  
        break;  
    }  
    case PROGRAM_INDICATOR:  
    {  
        Print(__FILE__+" is custom indicator");  
        break;  
    }  
    default:Print("MQL5 program type value is ",mql_program);  
}
```


Symbol Properties

To obtain the current market information there are several functions: [SymbolInfoInteger\(\)](#), [SymbolInfoDouble\(\)](#) and [SymbolInfoString\(\)](#). The first parameter is the symbol name, the values of the second function parameter can be one of the identifiers of ENUM_SYMBOL_INFO_INTEGER, ENUM_SYMBOL_INFO_DOUBLE and ENUM_SYMBOL_INFO_STRING.

For function [SymbolInfoInteger\(\)](#)

ENUM_SYMBOL_INFO_INTEGER

Identifier	Description	Type
SYMBOL_SELECT	Symbol is selected in Market Watch	bool
SYMBOL_SESSION_DEALS	Number of deals in the current session	long
SYMBOL_SESSION_BUY_ORDERS	Number of Buy orders at the moment	long
SYMBOL_SESSION_SELL_ORDERS	Number of Sell orders at the moment	long
SYMBOL_VOLUME	Volume of the last deal	long
SYMBOL_VOLUMEHIGH	Maximal day volume	long
SYMBOL_VOLUMELOW	Minimal day volume	long
SYMBOL_TIME	Time of the last quote	datetime
SYMBOL_DIGITS	Digits after a decimal point	int
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	bool
SYMBOL_SPREAD	Spread value in points	int
SYMBOL_TICKS_BOOKDEPTH	Maximal number of requests shown in Depth of Market . For symbols that have no queue of requests, the value is equal to zero.	int
SYMBOL_TRADE_CALC_MODE	Contract price calculation mode	ENUM_SYMBOL_CALC_MODE
SYMBOL_TRADE_MODE	Order execution type	ENUM_SYMBOL_TRADE_MODE
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	datetime
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	datetime
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to	int

	place Stop orders	
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	int
SYMBOL_TRADE_EXEMODE	Deal execution mode	ENUM_SYMBOL_TRADE_EXECUTION
SYMBOL_SWAP_MODE	Swap calculation model	ENUM_SYMBOL_SWAP_MODE
SYMBOL_SWAP_ROLLOVER3DAYS	Day of week to charge 3 days swap rollover	ENUM_DAY_OF_WEEK
SYMBOL_EXPIRATION_MODE	Flags of allowed order expiration modes	int
SYMBOL_FILLING_MODE	Flags of allowed order filling modes	int
SYMBOL_ORDER_MODE	Flags of allowed order types	int
SYMBOL_OPTION_MODE	Option type	ENUM_SYMBOL_OPTION_MODE
SYMBOL_OPTION_RIGHT	Option right (Call/Put)	ENUM_SYMBOL_OPTION_RIGHT

For function [SymbolInfoDouble\(\)](#)

ENUM_SYMBOL_INFO_DOUBLE

Identifier	Description	Type
SYMBOL_BID	Bid - best sell offer	double
SYMBOL_BIDHIGH	Maximal Bid of the day	double
SYMBOL_BIDLOW	Minimal Bid of the day	double
SYMBOL_ASK	Ask - best buy offer	double
SYMBOL_ASKHIGH	Maximal Ask of the day	double
SYMBOL_ASKLOW	Minimal Ask of the day	double
SYMBOL_LAST	Price of the last deal	double
SYMBOL_LASTHIGH	Maximal Last of the day	double
SYMBOL_LASTLOW	Minimal Last of the day	double
SYMBOL_OPTION_STRIKE	The strike price of an option. The price at which an option buyer can buy (in a Call option) or sell (in a Put option) the underlying asset, and the option seller is obliged to sell or buy the appropriate amount of the underlying asset.	double

SYMBOL_POINT	Symbol point value	double
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK_VALUE_PROFIT	double
SYMBOL_TRADE_TICK_VALUE_PROFIT	Calculated tick price for a profitable position	double
SYMBOL_TRADE_TICK_VALUE_LOSS	Calculated tick price for a losing position	double
SYMBOL_TRADE_TICK_SIZE	Minimal price change	double
SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	double
SYMBOL_VOLUME_MIN	Minimal volume for a deal	double
SYMBOL_VOLUME_MAX	Maximal volume for a deal	double
SYMBOL_VOLUME_STEP	Minimal volume change step for deal execution	double
SYMBOL_VOLUME_LIMIT	Maximum allowed aggregate volume of an open position and pending orders in one direction (buy or sell) for the symbol. For example, with the limitation of 5 lots, you can have an open buy position with the volume of 5 lots and place a pending order Sell Limit with the volume of 5 lots. But in this case you cannot place a Buy Limit pending order (since the total volume in one direction will exceed the limitation) or place Sell Limit with the volume more than 5 lots.	double
SYMBOL_SWAP_LONG	Long swap value	double
SYMBOL_SWAP_SHORT	Short swap value	double
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening a position with the volume of one lot. It is used for checking a client's assets when he or she enters the market.	double
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency	double

	of the symbol, charged from one lot. It is used for checking a client's assets when his/her account state changes. If the maintenance margin is equal to 0, the initial margin is used.	
SYMBOL_SESSION_VOLUME	Summary volume of current session deals	double
SYMBOL_SESSION_TURNOVER	Summary turnover of the current session	double
SYMBOL_SESSION_INTEREST	Summary open interest	double
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Current volume of Buy orders	double
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Current volume of Sell orders	double
SYMBOL_SESSION_OPEN	Open price of the current session	double
SYMBOL_SESSION_CLOSE	Close price of the current session	double
SYMBOL_SESSION_AW	Average weighted price of the current session	double
SYMBOL_SESSION_PRICE_SETTLEMENT	Settlement price of the current session	double
SYMBOL_SESSION_PRICE_LIMIT_MIN	Minimal price of the current session	double
SYMBOL_SESSION_PRICE_LIMIT_MAX	Maximal price of the current session	double
SYMBOL_MARGIN_HEDGED	<p>Contract size or margin value per one lot of hedged positions (oppositely directed positions of one symbol). Two margin calculation methods are possible for hedged positions. The calculation method is defined by the broker.</p> <p>Basic calculation:</p> <ul style="list-style-type: none"> • If the initial margin (SYMBOL_MARGIN_INITIAL) is specified for a symbol, the hedged margin is specified as an absolute value (in monetary terms). 	double

	<ul style="list-style-type: none"> • If the initial margin is not specified (equal to 0), SYMBOL_MARGIN_HEDGED is equal to the size of the contract, that will be used to calculate the margin by the appropriate formula in accordance with the type of the financial instrument (SYMBOL_TRADE_CALC_MODE). <p>Calculation for the largest position:</p> <ul style="list-style-type: none"> • The SYMBOL_MARGIN_HEDGED value is not taken into account. • The volume of all short and all long positions of a symbol is calculated. • For each direction, a weighted average open price and a weighted average rate of conversion to the deposit currency is calculated. • Next, using the appropriate formula chosen in accordance with the symbol type (SYMBOL_TRADE_CALC_MODE) the margin is calculate for the shot and the long part. • The largest one of the values is used as the margin. 	
--	--	--

For function [SymbolInfoString\(\)](#)

ENUM_SYMBOL_INFO_STRING

Identifier	Description	Type
SYMBOL_BASIS	The underlying asset of a derivative	string
SYMBOL_CURRENCY_BASE	Basic currency of a symbol	string
SYMBOL_CURRENCY_PROFIT	Profit currency	string
SYMBOL_CURRENCY_MARGIN	Margin currency	string
SYMBOL_BANK	Feeder of the current quote	string
SYMBOL_DESCRIPTION	Symbol description	string

SYMBOL_ISIN	The name of a symbol in the ISIN system (International Securities Identification Number). The International Securities Identification Number is a 12-digit alphanumeric code that uniquely identifies a security. The presence of this symbol property is determined on the side of a trade server.	string
SYMBOL_PATH	Path in the symbol tree	string

For each symbol several expiration modes of pending orders can be specified. A flag is matched to each mode. Flags can be combined using the operation of logical **OR** (`|`), for example, `SYMBOL_EXPIRATION_GTC|SYMBOL_EXPIRATION_SPECIFIED`. In order to check whether a certain mode is allowed for the symbol, the result of the logical **AND** (`&`) should be compared to the mode flag.

If flag `SYMBOL_EXPIRATION_SPECIFIED` is specified for a symbol, then while sending a pending order, you may specify the moment this pending order is valid till.

Identifier	Value	Description
SYMBOL_EXPIRATION_GTC	1	The order is valid during the unlimited time period, until it is explicitly canceled
SYMBOL_EXPIRATION_DAY	2	The order is valid till the end of the day
SYMBOL_EXPIRATION_SPECIFIED	4	The expiration time is specified in the order
SYMBOL_EXPIRATION_SPECIFIED_DAY	8	The expiration date is specified in the order

Example:

```
//+-----+
//| Checks if the specified expiration mode is allowed |
//+-----+
bool IsExpirationTypeAllowed(string symbol,int exp_type)
{
    //-- Obtain the value of the property that describes allowed expiration modes
    int expiration=(int)SymbolInfoInteger(symbol,SYMBOL_EXPIRATION_MODE);
    //-- Return true, if mode exp_type is allowed
    return((expiration&exp_type)==exp_type);
}
```


When sending an order, you can specify the filling policy for the volume set in the order. Allowed order filling modes for each symbol are specified in the table. You can set several modes for one symbol by combining flags. The flags can be combined by the operation of the logical **OR** (`|`), for example, `SYMBOL_FILLING_FOK|SYMBOL_FILLING_IOC`. In order to check whether a certain mode is allowed for the symbol, the result of the logical **AND** (`&`) should be compared to the mode flag.

Fill Policy	Identifier	Value	Description
Fill or Kill	<code>SYMBOL_FILLING_FOK</code>	1	This policy means that a deal can be executed only with the specified volume. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The required volume can be filled using several offers available on the market at the moment.
Immediate or Cancel	<code>SYMBOL_FILLING_IOC</code>	2	In this case a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case the order cannot be filled completely, the available volume of the order will be filled, and the remaining volume will be canceled. The possibility of using IOC orders is determined at the trade server.
Return	No identifier		This policy is used only for market orders (Buy and Sell), limit and stop limit orders and only for the symbols with Market or Exchange execution. In case of partial filling a market or limit order with remaining volume

			is not canceled but processed further.
--	--	--	--

In the Request and Instant [execution modes](#) the Fill or Kill policy is always used for market orders, and the Return policy is always used for limit orders. In this case, when sending orders using [OrderSend](#) or [OrderSendAsync](#), there is no need to specify a fill policy for them.

In the Market and Exchange execution modes the Return policy is always allowed for all the order types. To find out whether the other policies are allowed, use the SYMBOL_FILLING_FOK and SYMBOL_FILLING_IOC properties.

Example:

```
//+-----+
//| Checks if the specified filling mode is allowed |
//+-----+
bool IsFillingTypeAllowed(string symbol,int fill_type)
{
    //-- Obtain the value of the property that describes allowed filling modes
    int filling=(int)SymbolInfoInteger(symbol,SYMBOL_FILLING_MODE);
    //-- Return true, if mode fill_type is allowed
    return((filling & fill_type)==fill_type);
}
```

When sending a [trade request](#) using OrderSend() function, an order type from [ENUM_ORDER_TYPE enumeration](#) should be specified for some operations. Not all types of orders may be allowed for a specific symbol. [SYMBOL_ORDER_MODE](#) property describes the flags of the allowed order types.

Identifier	Value	Description
SYMBOL_ORDER_MARKET	1	Market orders are allowed (Buy and Sell)
SYMBOL_ORDER_LIMIT	2	Limit orders are allowed (Buy Limit and Sell Limit)
SYMBOL_ORDER_STOP	4	Stop orders are allowed (Buy Stop and Sell Stop)
SYMBOL_ORDER_STOP_LIMIT	8	Stop-limit orders are allowed (Buy Stop Limit and Sell Stop Limit)
SYMBOL_ORDER_SL	16	Stop Loss is allowed
SYMBOL_ORDER_TP	32	Take Profit is allowed

Example:

```
//+-----+
//| The function prints out order types allowed for a symbol |
//+-----+
```



```

void Check_SYMBOL_ORDER_MODE(string symbol)
{
//--- receive the value of the property describing allowed order types
    int symbol_order_mode=(int)SymbolInfoInteger(symbol,SYMBOL_ORDER_MODE);
//--- check for market orders (Market Execution)
    if((SYMBOL_ORDER_MARKET&symbol_order_mode)==SYMBOL_ORDER_MARKET)
        Print(symbol+": Market orders are allowed (Buy and Sell)");
//--- check for Limit orders
    if((SYMBOL_ORDER_LIMIT&symbol_order_mode)==SYMBOL_ORDER_LIMIT)
        Print(symbol+": Buy Limit and Sell Limit orders are allowed");
//--- check for Stop orders
    if((SYMBOL_ORDER_STOP&symbol_order_mode)==SYMBOL_ORDER_STOP)
        Print(symbol+": Buy Stop and Sell Stop orders are allowed");
//--- check for Stop Limit orders
    if((SYMBOL_ORDER_STOP_LIMIT&symbol_order_mode)==SYMBOL_ORDER_STOP_LIMIT)
        Print(symbol+": Buy Stop Limit and Sell Stop Limit orders are allowed");
//--- check if placing a Stop Loss orders is allowed
    if((SYMBOL_ORDER_SL&symbol_order_mode)==SYMBOL_ORDER_SL)
        Print(symbol+": Stop Loss orders are allowed");
//--- check if placing a Take Profit orders is allowed
    if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_TP)
        Print(symbol+": Take Profit orders are allowed");
//---
}

```

The ENUM_SYMBOL_CALC_MODE enumeration is used for obtaining information about how the margin requirements for a symbol are calculated.

ENUM_SYMBOL_CALC_MODE

Identifier	Description	Formula
SYMBOL_CALC_MODE_FOREX	Forex mode - calculation of profit and margin for Forex	Margin: Lots*Contract_Size/Leverage Profit: (close_price-open_price) *Contract_Size*Lots
SYMBOL_CALC_MODE_FUTURE S	Futures mode - calculation of margin and profit for futures	Margin: Lots *InitialMargin*Percentage/100 Profit: (close_price-open_price) *TickPrice/TickSize*Lots
SYMBOL_CALC_MODE_CFD	CFD mode - calculation of margin and profit for CFD	Margin: Lots *ContractSize*MarketPrice*Percentage/100 Profit: (close_price-open_price) *Contract_Size*Lots

SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - calculation of margin and profit for CFD by indexes	Margin: $(Lots * ContractSize * MarketPrice) * TickPrice / TickSize$ Profit: $(close_price - open_price) * Contract_Size * Lots$
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - calculation of margin and profit for CFD at leverage trading	Margin: $(Lots * ContractSize * MarketPrice * Percentage) / Leverage$ Profit: $(close_price - open_price) * Contract_Size * Lots$
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode - calculation of margin and profit for trading securities on a stock exchange	Margin: $Lots * ContractSize * OpenPrice$ Profit: $(close_price - open_price) * Contract_Size * Lots$
SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - calculation of margin and profit for trading futures contracts on a stock exchange	Margin: $Lots * InitialMargin$ or $Lots * MaintenanceMargin$ Profit: $(close_price - open_price) * Lots * TickPrice / TickSize$
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode - calculation of margin and profit for trading futures contracts on FORTS. The margin may be reduced by the amount of MarginDiscount deviation according to the following rules: 1. If the price of a long position (buy order) is less than the estimated price, $MarginDiscount = Lots * ((PriceSettle - PriceOrder) * TickPrice / TickSize)$ 2. If the price of a short position (sell order) exceeds the estimated price, $MarginDiscount = Lots * ((PriceOrder - PriceSettle) * TickPrice / TickSize)$ where: <ul style="list-style-type: none"> PriceSettle - estimated (clearing) price of the previous session; PriceOrder - average weighted position price or open price set in the order (request); 	Margin: $Lots * InitialMargin$ or $Lots * MaintenanceMargin$ Profit: $(close_price - open_price) * Lots * TickPrice / TickSize$

	<ul style="list-style-type: none"> ○ TickPrice - tick price (cost of the price change by one point) ○ TickSize - tick size (minimum price change step) 	
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - a symbol is used as a non-tradable asset on a trading account. The market value of an open position is calculated based on the volume, current market price, contract size and liquidity ratio. The value is included into Assets, which are added to Equity. Open positions of such symbols increase the Free Margin amount and are used as additional margin (collateral) for open positions of tradable instruments.	Margin: no Profit: no Market Value: $\text{Lots} * \text{ContractSize} * \text{MarketPrice} * \text{LiquidityRate}$

There are several symbol trading modes. Information about trading modes of a certain symbol is reflected in the values of enumeration `ENUM_SYMBOL_TRADE_MODE`.

ENUM_SYMBOL_TRADE_MODE

Identifier	Description
SYMBOL_TRADE_MODE_DISABLED	Trade is disabled for the symbol
SYMBOL_TRADE_MODE_LONGONLY	Allowed only long positions
SYMBOL_TRADE_MODE_SHORTONLY	Allowed only short positions
SYMBOL_TRADE_MODE_CLOSEONLY	Allowed only position close operations
SYMBOL_TRADE_MODE_FULL	No trade restrictions

Possible deal execution modes for a certain symbol are defined in enumeration `ENUM_SYMBOL_TRADE_EXECUTION`.

ENUM_SYMBOL_TRADE_EXECUTION

Identifier	Description
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution

SYMBOL_TRADE_EXECUTION_MARKET	Market execution
SYMBOL_TRADE_EXECUTION_EXCHANGE	Exchange execution

Methods of swap calculation at position transfer are specified in enumeration `ENUM_SYMBOL_SWAP_MODE`. The method of swap calculation determines the units of measure of the [SYMBOL_SWAP_LONG](#) and [SYMBOL_SWAP_SHORT](#) parameters. For example, if swaps are charged in the client deposit currency, then the values of those parameters are specified as an amount of money in the client deposit currency.

ENUM_SYMBOL_SWAP_MODE

Identifier	Description
SYMBOL_SWAP_MODE_DISABLED	Swaps disabled (no swaps)
SYMBOL_SWAP_MODE_POINTS	Swaps are charged in points
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Swaps are charged in money in base currency of the symbol
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Swaps are charged in money in margin currency of the symbol
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Swaps are charged in money, in client deposit currency
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Swaps are charged as the specified annual interest from the instrument price at calculation of swap (standard bank year is 360 days)
SYMBOL_SWAP_MODE_INTEREST_OPEN	Swaps are charged as the specified annual interest from the open price of position (standard bank year is 360 days)
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the close price +/- specified number of points (parameters <code>SYMBOL_SWAP_LONG</code> and <code>SYMBOL_SWAP_SHORT</code>)
SYMBOL_SWAP_MODE_REOPEN_BID	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the current Bid price +/- specified number of points (parameters <code>SYMBOL_SWAP_LONG</code> and <code>SYMBOL_SWAP_SHORT</code>)

Values of the `ENUM_DAY_OF_WEEK` enumeration are used for specifying days of week.

ENUM_DAY_OF_WEEK

Identifier	Description
SUNDAY	Sunday
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday

An option is a contract, which gives the right, but not the obligation, to buy or sell an underlying asset (goods, stocks, futures, etc.) at a specified price on or before a specific date. The following enumerations describe option properties, including the option type and the right arising from it.

ENUM_SYMBOL_OPTION_RIGHT

Identifier	Description
SYMBOL_OPTION_RIGHT_CALL	A call option gives you the right to buy an asset at a specified price
SYMBOL_OPTION_RIGHT_PUT	A put option gives you the right to sell an asset at a specified price

ENUM_SYMBOL_OPTION_MODE

Identifier	Description
SYMBOL_OPTION_MODE_EUROPEAN	European option may only be exercised on a specified date (expiration, execution date, delivery date)
SYMBOL_OPTION_MODE_AMERICAN	American option may be exercised on any trading day on or before expiry. The period within which a buyer can exercise the option is specified for it

Account Properties

To obtain information about the current account there are several functions: [AccountInfoInteger\(\)](#), [AccountInfoDouble\(\)](#) and [AccountInfoString\(\)](#). The function parameter values can accept values from the corresponding ENUM_ACCOUNT_INFO enumerations.

For the function [AccountInfoInteger\(\)](#)

ENUM_ACCOUNT_INFO_INTEGER

Identifier	Description	Type
ACCOUNT_LOGIN	Account number	long
ACCOUNT_TRADE_MODE	Account trade mode	ENUM_ACCOUNT_TRADE_MODE
ACCOUNT_LEVERAGE	Account leverage	long
ACCOUNT_LIMIT_ORDERS	Maximum allowed number of active pending orders	int
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	ENUM_ACCOUNT_STOPOUT_MODE
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	bool
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert Advisor	bool
ACCOUNT_MARGIN_MODE	Margin calculation mode	ENUM_ACCOUNT_MARGIN_MODE

For the function [AccountInfoDouble\(\)](#)

ENUM_ACCOUNT_INFO_DOUBLE

Identifier	Description	Type
ACCOUNT_BALANCE	Account balance in the deposit currency	double
ACCOUNT_CREDIT	Account credit in the deposit currency	double
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	double
ACCOUNT_EQUITY	Account equity in the deposit currency	double
ACCOUNT_MARGIN	Account margin used in the deposit currency	double
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	double

ACCOUNT_MARGIN_LEVEL	Account margin level in percents	double
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double
ACCOUNT_MARGIN_INITIAL	Initial margin. The amount reserved on an account to cover the margin of all pending orders	double
ACCOUNT_MARGIN_MAINTENANCE	Maintenance margin. The minimum equity reserved on an account to cover the minimum amount of all open positions	double
ACCOUNT_ASSETS	The current assets of an account	double
ACCOUNT_LIABILITIES	The current liabilities on an account	double
ACCOUNT_COMMISSION_BLOCKED	The current blocked commission amount on an account	double

For function [AccountInfoString\(\)](#)

ENUM_ACCOUNT_INFO_STRING

Identifier	Description	Type
ACCOUNT_NAME	Client name	string
ACCOUNT_SERVER	Trade server name	string
ACCOUNT_CURRENCY	Account currency	string
ACCOUNT_COMPANY	Name of a company that serves the account	string

There are several types of accounts that can be opened on a trade server. The type of account on which an MQL5 program is running can be found out using the ENUM_ACCOUNT_TRADE_MODE enumeration.

ENUM_ACCOUNT_TRADE_MODE

Identifier	Description
ACCOUNT_TRADE_MODE_DEMO	Demo account
ACCOUNT_TRADE_MODE_CONTEST	Contest account
ACCOUNT_TRADE_MODE_REAL	Real account

In case equity is not enough for maintaining open positions, the Stop Out situation, i.e. forced closing occurs. The minimum margin level at which Stop Out occurs can be set in percentage or in monetary terms. To find out the mode set for the account use the `ENUM_ACCOUNT_STOPOUT_MODE` enumeration.

ENUM_ACCOUNT_STOPOUT_MODE

Identifier	Description
ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money

ENUM_ACCOUNT_MARGIN_MODE

Identifier	Description
ACCOUNT_MARGIN_MODE_RETAIL_NETTING	Used for the OTC markets to interpret positions in the "netting" mode (only one position can exist for one symbol). The margin is calculated based on the symbol type (SYMBOL_TRADE_CALC_MODE).
ACCOUNT_MARGIN_MODE_EXCHANGE	Used for the exchange markets. Margin is calculated based on the discounts specified in symbol settings. Discounts are set by the broker, but not less than the values set by the exchange.
ACCOUNT_MARGIN_MODE_RETAIL_HEDGING	Used for the exchange markets where individual positions are possible (hedging, multiple positions can exist for one symbol). The margin is calculated based on the symbol type (SYMBOL_TRADE_CALC_MODE) taking into account the hedged margin (SYMBOL_MARGIN_HEDGED).

An example of the script that outputs a brief account information.

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //-- Name of the company
    string company=AccountInfoString(ACCOUNT_COMPANY);
    //-- Name of the client
```



```

    string name=AccountInfoString(ACCOUNT_NAME);
//--- Account number
    long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- Name of the server
    string server=AccountInfoString(ACCOUNT_SERVER);
//--- Account currency
    string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- Demo, contest or real account
    ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(AC
//--- Now transform the value of the enumeration into an understandable form
    string trade_mode;
    switch(account_type)
    {
        case ACCOUNT_TRADE_MODE_DEMO:
            trade_mode="demo";
            break;
        case ACCOUNT_TRADE_MODE_CONTEST:
            trade_mode="contest";
            break;
        default:
            trade_mode="real";
            break;
    }
//--- Stop Out is set in percentage or money
    ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteg
//--- Get the value of the levels when Margin Call and Stop Out occur
    double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
    double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- Show brief account information
    PrintFormat("The account of the client '%s' #d %s opened in '%s' on the server '%s'
                name,login,trade_mode,company,server);
    PrintFormat("Account currency - %s, MarginCall and StopOut levels are set in %s",
                currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"percentage":"r
    PrintFormat("MarginCall=%G, StopOut=%G",margin_call,stop_out);
}

```


Testing Statistics

After the testing is over, different parameters of the trading results statistics are calculated. The values of the parameters can be obtained using the [TesterStatistics\(\)](#) function, by specifying the parameter ID from the ENUM_STATISTICS enumeration.

Although two types of parameters (int and double) are used for calculating statistics, the function returns all values in the double form. All the statistic values of the double type are expressed in the deposit currency by default, unless otherwise specified.

ENUM_STATISTICS

ID	Description of a statistic parameter	Type
STAT_INITIAL_DEPOSIT	The value of the initial deposit	double
STAT_WITHDRAWAL	Money withdrawn from an account	double
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS (STAT_GROSS_LOSS is always less than or equal to zero)	double
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal to zero	double
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	double
STAT_MAX_PROFITTRADE	Maximum profit - the largest value of all profitable trades. The value is greater than or equal to zero	double
STAT_MAX_LOSSTRADE	Maximum loss - the lowest value of all losing trades. The value is less than or equal to zero	double
STAT_CONPROFITMAX	Maximum profit in a series of profitable trades. The value is greater than or equal to zero	double
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	int

STAT_MAX_CONWINS	The total profit of the longest series of profitable trades	double
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	int
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	double
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	int
STAT_MAX_CONLOSSES	The total loss of the longest series of losing trades	double
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	int
STAT_BALANCEMIN	Minimum balance value	double
STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous drawdowns; here the largest value is taken	double
STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	double
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	double

STAT_EQUITYMIN	Minimum equity value	double
STAT_EQUITY_DD	Maximum equity drawdown in monetary terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	double
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity drawdown in monetary terms (STAT_EQUITY_DD).	double
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	double
STAT_EXPECTED_PAYOFF	Expected payoff	double
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of $\text{STAT_GROSS_PROFIT} / \text{STAT_GROSS_LOSS}$. If $\text{STAT_GROSS_LOSS}=0$, the profit factor is equal to DBL_MAX	double
STAT_RECOVERY_FACTOR	Recovery factor, equal to the ratio of $\text{STAT_PROFIT} / \text{STAT_BALANCE_DD}$	double
STAT_SHARPE_RATIO	Sharpe ratio	double
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	double
STAT_CUSTOM_ONTESTER	The value of the calculated custom optimization criterion returned by the OnTester() function	double

STAT_DEALS	The number of deals	int
STAT_TRADES	The number of trades	int
STAT_PROFIT_TRADES	Profitable trades	int
STAT_LOSS_TRADES	Losing trades	int
STAT_SHORT_TRADES	Short trades	int
STAT_LONG_TRADES	Long trades	int
STAT_PROFIT_SHORTTRADES	Profitable short trades	int
STAT_PROFIT_LONGTRADES	Profitable long trades	int
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	int
STAT_LOSSTRADES_AVGCON	Average length of a losing series of trades	int

Trade Constants

Various constants used for programming trading strategies are divided into the following groups:

- [History Database Properties](#) - receiving general information on a symbol;
- [Order properties](#) - obtaining information about trade orders;
- [Position properties](#) - obtaining information about current positions;
- [Deal properties](#) - obtaining information about deals;
- [Trade operation types](#) - description of trade operations available;
- [Trade transaction types](#) - description of possible trade transactions types;
- [Trade orders in DOM](#) - separation of orders according to the direction of a requested operation.

History Database Properties

When accessing [timeseries](#) the [SeriesInfoInteger\(\)](#) function is used for obtaining additional [symbol information](#). Identifier of a required property is passed as the function parameter. The identifier can be one of values of ENUM_SERIES_INFO_INTEGER.

ENUM_SERIES_INFO_INTEGER

Identifier	Description	Type
SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	long
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	datetime
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	datetime
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	datetime
SERIES_TERMINAL_FIRSTDATE	The very first date in the history of the symbol in the client terminal, regardless of the timeframe	datetime
SERIES_SYNCHRONIZED	Symbol/period data synchronization flag for the current moment	bool

Order Properties

Requests to execute trade operations are formalized as orders. Each order has a variety of properties for reading. Information on them can be obtained using functions [OrderGet...\(\)](#) and [HistoryOrderGet...\(\)](#).

For functions [OrderGetInteger\(\)](#) and [HistoryOrderGetInteger\(\)](#)

ENUM_ORDER_PROPERTY_INTEGER

Identifier	Description	Type
ORDER_TICKET	Order ticket. Unique number assigned to each order	long
ORDER_TIME_SETUP	Order setup time	datetime
ORDER_TYPE	Order type	ENUM_ORDER_TYPE
ORDER_STATE	Order state	ENUM_ORDER_STATE
ORDER_TIME_EXPIRATION	Order expiration time	datetime
ORDER_TIME_DONE	Order execution or cancellation time	datetime
ORDER_TIME_SETUP_MSC	The time of placing an order for execution in milliseconds since 01.01.1970	long
ORDER_TIME_DONE_MSC	Order execution/cancellation time in milliseconds since 01.01.1970	long
ORDER_TYPE_FILLING	Order filling type	ENUM_ORDER_TYPE_FILLING
ORDER_TYPE_TIME	Order lifetime	ENUM_ORDER_TYPE_TIME
ORDER_MAGIC	ID of an Expert Advisor that has placed the order (designed to ensure that each Expert Advisor places its own unique number)	long
ORDER_POSITION_ID	Position identifier that is set to an order as soon as it is executed. Each executed order results in a deal that opens or modifies an already existing position. The identifier of exactly this position is set to the executed order at this moment.	long
ORDER_POSITION_BY_ID	Identifier of an opposite position used for closing by	long

	order ORDER_TYPE_CLOSE_BY	
--	---------------------------	--

For functions [OrderGetDouble\(\)](#) and [HistoryOrderGetDouble\(\)](#)

ENUM_ORDER_PROPERTY_DOUBLE

Identifier	Description	Type
ORDER_VOLUME_INITIAL	Order initial volume	double
ORDER_VOLUME_CURRENT	Order current volume	double
ORDER_PRICE_OPEN	Price specified in the order	double
ORDER_SL	Stop Loss value	double
ORDER_TP	Take Profit value	double
ORDER_PRICE_CURRENT	The current price of the order symbol	double
ORDER_PRICE_STOPLIMIT	The Limit order price for the StopLimit order	double

For functions [OrderGetString\(\)](#) and [HistoryOrderGetString\(\)](#)

ENUM_ORDER_PROPERTY_STRING

Identifier	Description	Type
ORDER_SYMBOL	Symbol of the order	string
ORDER_COMMENT	Order comment	string
ORDER_EXTERNAL_ID	Order identifier in an external trading system (on the Exchange)	string

When sending a trade request using the [OrderSend\(\)](#) function, some operations require the indication of the order type. The order type is specified in the *type* field of the special structure [MqlTradeRequest](#), and can accept values of the ENUM_ORDER_TYPE enumeration.

ENUM_ORDER_TYPE

Identifier	Description
ORDER_TYPE_BUY	Market Buy order
ORDER_TYPE_SELL	Market Sell order
ORDER_TYPE_BUY_LIMIT	Buy Limit pending order
ORDER_TYPE_SELL_LIMIT	Sell Limit pending order
ORDER_TYPE_BUY_STOP	Buy Stop pending order

ORDER_TYPE_SELL_STOP	Sell Stop pending order
ORDER_TYPE_BUY_STOP_LIMIT	Upon reaching the order price, a pending Buy Limit order is placed at the StopLimit price
ORDER_TYPE_SELL_STOP_LIMIT	Upon reaching the order price, a pending Sell Limit order is placed at the StopLimit price
ORDER_TYPE_CLOSE_BY	Order to close a position by an opposite one

Each order has a status that describes its state. To obtain information, use [OrderGetInteger\(\)](#) or [HistoryOrderGetInteger\(\)](#) with the ORDER_STATE modifier. Allowed values are stored in the ENUM_ORDER_STATE enumeration.

ENUM_ORDER_STATE

Identifier	Description
ORDER_STATE_STARTED	Order checked, but not yet accepted by broker
ORDER_STATE_PLACED	Order accepted
ORDER_STATE_CANCELED	Order canceled by client
ORDER_STATE_PARTIAL	Order partially executed
ORDER_STATE_FILLED	Order fully executed
ORDER_STATE_REJECTED	Order rejected
ORDER_STATE_EXPIRED	Order expired
ORDER_STATE_REQUEST_ADD	Order is being registered (placing to the trading system)
ORDER_STATE_REQUEST_MODIFY	Order is being modified (changing its parameters)
ORDER_STATE_REQUEST_CANCEL	Order is being deleted (deleting from the trading system)

When sending a trade request using the [OrderSend\(\)](#) function, the filling policy can be set for an order in the *type_filling* field of the special structure [MqlTradeRequest](#). Values of the ENUM_ORDER_TYPE_FILLING enumeration are allowed. To obtain the value of this property, use the function [OrderGetInteger\(\)](#) or [HistoryOrderGetInteger\(\)](#) with the ORDER_TYPE_FILLING modifier.

ENUM_ORDER_TYPE_FILLING

Identifier	Description
ORDER_FILLING_FOK	This filling policy means that an order can be filled only in the specified amount. If the necessary amount of a financial instrument is

	currently unavailable in the market, the order will not be executed. The required volume can be filled using several offers available on the market at the moment.
ORDER_FILLING_IOC	This mode means that a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case the the entire volume of an order cannot be filled, the available volume of it will be filled, and the remaining volume will be canceled.
ORDER_FILLING_RETURN	This policy is used only for market orders (ORDER_TYPE_BUY and ORDER_TYPE_SELL), limit and stop limit orders (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT) and only for the symbols with Market or Exchange execution . In case of partial filling a market or limit order with remaining volume is not canceled but processed further. For the activation of the ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT orders, a corresponding limit order ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT with the ORDER_FILLING_RETURN execution type is created.

The order validity period can be set in the *type_time* field of the special structure [MqlTradeRequest](#) when sending a trade request using the [OrderSend\(\)](#) function. Values of the ENUM_ORDER_TYPE_TIME enumeration are allowed. To obtain the value of this property use the function [OrderGetInteger\(\)](#) or [HistoryOrderGetInteger\(\)](#) with the ORDER_TYPE_TIME modifier.

ENUM_ORDER_TYPE_TIME

Identifier	Description
ORDER_TIME_GTC	Good till cancel order
ORDER_TIME_DAY	Good till current trade day order
ORDER_TIME_SPECIFIED	Good till expired order
ORDER_TIME_SPECIFIED_DAY	The order will be effective till 23:59:59 of the specified day. If this time is outside a trading session, the order expires in the nearest trading time.

Position Properties

Execution of [trade operations](#) results in the opening of a position, changing of its volume and/or direction, or its disappearance. Trade operations are conducted based on [orders](#), sent by the [OrderSend\(\)](#) function in the form of [trade requests](#). For each financial [security](#) (symbol) only one open position is possible. A position has a set of properties available for reading by the [PositionGet...\(\)](#) functions.

For the function [PositionGetInteger\(\)](#)

ENUM_POSITION_PROPERTY_INTEGER

Identifier	Description	Type
POSITION_TICKET	Position ticket. Unique number assigned to each newly opened position. It usually matches the ticket of an order used to open the position except when the ticket is changed as a result of service operations on the server, for example, when charging swaps with position re-opening. To find an order used to open a position, apply the POSITION_IDENTIFIER property. POSITION_TICKET value corresponds to MqlTradeRequest::position .	long
POSITION_TIME	Position open time	datetime
POSITION_TIME_MSC	Position opening time in milliseconds since 01.01.1970	long
POSITION_TIME_UPDATE	Position changing time in seconds since 01.01.1970	long
POSITION_TIME_UPDATE_MSC	Position changing time in milliseconds since 01.01.1970	long
POSITION_TYPE	Position type	ENUM_POSITION_TYPE
POSITION_MAGIC	Position magic number (see ORDER_MAGIC)	long
POSITION_IDENTIFIER	Position identifier is a unique number assigned to each re-opened position. It does not change throughout its life cycle and corresponds to the ticket	long

	<p>of an order used to open a position.</p> <p>Position identifier is specified in each order (ORDER_POSITION_ID) and deal (DEAL_POSITION_ID) used to open, modify, or close it. Use this property to search for orders and deals related to the position.</p> <p>When reversing a position in netting mode (using a single in/out trade), POSITION_IDENTIFIER does not change. However, POSITION_TICKET is replaced with the ticket of the order that led to the reversal. Position reversal is not provided in hedging mode.</p>	
--	--	--

For the function [PositionGetDouble\(\)](#)

ENUM_POSITION_PROPERTY_DOUBLE

Identifier	Description	Type
POSITION_VOLUME	Position volume	double
POSITION_PRICE_OPEN	Position open price	double
POSITION_SL	Stop Loss level of opened position	double
POSITION_TP	Take Profit level of opened position	double
POSITION_PRICE_CURRENT	Current price of the position symbol	double
POSITION_SWAP	Cumulative swap	double
POSITION_PROFIT	Current profit	double

For the function [PositionGetString\(\)](#)

ENUM_POSITION_PROPERTY_STRING

Identifier	Description	Type
POSITION_SYMBOL	Symbol of the position	string
POSITION_COMMENT	Position comment	string

Direction of an open position (buy or sell) is defined by the value from the ENUM_POSITION_TYPE enumeration. In order to obtain the type of an open position use the [PositionGetInteger\(\)](#) function with the POSITION_TYPE modifier.

ENUM_POSITION_TYPE

Identifier	Description
POSITION_TYPE_BUY	Buy
POSITION_TYPE_SELL	Sell

Deal Properties

A deal is the reflection of the fact of a [trade operation](#) execution based on an [order](#) that contains a trade request. Each trade is described by properties that allow to obtain information about it. In order to read values of properties, functions of the [HistoryDealGet...\(\)](#) type are used, that return values from corresponding enumerations.

For the function [HistoryDealGetInteger\(\)](#)

ENUM_DEAL_PROPERTY_INTEGER

Identifier	Description	Type
DEAL_TICKET	Deal ticket. Unique number assigned to each deal	long
DEAL_ORDER	Deal order number	long
DEAL_TIME	Deal time	datetime
DEAL_TIME_MSC	The time of a deal execution in milliseconds since 01.01.1970	long
DEAL_TYPE	Deal type	ENUM_DEAL_TYPE
DEAL_ENTRY	Deal entry - entry in, entry out, reverse	ENUM_DEAL_ENTRY
DEAL_MAGIC	Deal magic number (see ORDER_MAGIC)	long
DEAL_POSITION_ID	Identifier of a position , in the opening, modification or closing of which this deal took part. Each position has a unique identifier that is assigned to all deals executed for the symbol during the entire lifetime of the position.	long

For the function [HistoryDealGetDouble\(\)](#)

ENUM_DEAL_PROPERTY_DOUBLE

Identifier	Description	Type
DEAL_VOLUME	Deal volume	double
DEAL_PRICE	Deal price	double
DEAL_COMMISSION	Deal commission	double
DEAL_SWAP	Cumulative swap on close	double
DEAL_PROFIT	Deal profit	double

For the function [HistoryDealGetString\(\)](#)

ENUM_DEAL_PROPERTY_STRING

Identifier	Description	Type
DEAL_SYMBOL	Deal symbol	string
DEAL_COMMENT	Deal comment	string
DEAL_EXTERNAL_ID	Deal identifier in an external trading system (on the Exchange)	string

Each deal is characterized by a type, allowed values are enumerated in ENUM_DEAL_TYPE. In order to obtain information about the deal type, use the [HistoryDealGetInteger\(\)](#) function with the DEAL_TYPE modifier.

ENUM_DEAL_TYPE

Identifier	Description
DEAL_TYPE_BUY	Buy
DEAL_TYPE_SELL	Sell
DEAL_TYPE_BALANCE	Balance
DEAL_TYPE_CREDIT	Credit
DEAL_TYPE_CHARGE	Additional charge
DEAL_TYPE_CORRECTION	Correction
DEAL_TYPE_BONUS	Bonus
DEAL_TYPE_COMMISSION	Additional commission
DEAL_TYPE_COMMISSION_DAILY	Daily commission
DEAL_TYPE_COMMISSION_MONTHLY	Monthly commission
DEAL_TYPE_COMMISSION_AGENT_DAILY	Daily agent commission
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Monthly agent commission
DEAL_TYPE_INTEREST	Interest rate
DEAL_TYPE_BUY_CANCELED	Canceled buy deal. There can be a situation when a previously executed buy deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_BUY) is changed to DEAL_TYPE_BUY_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation

DEAL_TYPE_SELL_CANCELED	Canceled sell deal. There can be a situation when a previously executed sell deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_SELL) is changed to DEAL_TYPE_SELL_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation
DEAL_DIVIDEND	Dividend operations
DEAL_DIVIDEND_FRANKED	Franked (non-taxable) dividend operations
DEAL_TAX	Tax charges

Deals differ not only in their types set in ENUM_DEAL_TYPE, but also in the way they change positions. This can be a simple position opening, or accumulation of a previously opened position (market entering), position closing by an opposite deal of a corresponding volume (market exiting), or position reversing, if the opposite-direction deal covers the volume of the previously opened position.

All these situations are described by values from the ENUM_DEAL_ENTRY enumeration. In order to receive this information about a deal, use the [HistoryDealGetInteger\(\)](#) function with the DEAL_ENTRY modifier.

ENUM_DEAL_ENTRY

Identifier	Description
DEAL_ENTRY_IN	Entry in
DEAL_ENTRY_OUT	Entry out
DEAL_ENTRY_INOUT	Reverse
DEAL_ENTRY_OUT_BY	Close a position by an opposite one

Trade Operation Types

Trading is done by sending orders to open positions using the [OrderSend\(\)](#) function, as well as to place, modify or delete pending orders. Each trade order refers to the type of the requested operation. Trading operations are described in the ENUM_TRADE_REQUEST_ACTIONS enumeration.

ENUM_TRADE_REQUEST_ACTIONS

Identifier	Description
TRADE_ACTION_DEAL	Place a trade order for an immediate execution with the specified parameters (market order)
TRADE_ACTION_PENDING	Place a trade order for the execution under specified conditions (pending order)
TRADE_ACTION_SLTP	Modify Stop Loss and Take Profit values of an opened position
TRADE_ACTION_MODIFY	Modify the parameters of the order placed previously
TRADE_ACTION_REMOVE	Delete the pending order placed previously
TRADE_ACTION_CLOSE_BY	Close a position by an opposite one

Example of the [TRADE_ACTION_DEAL](#) trade operation for opening a Buy position:

```
#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Opening Buy position |
//+-----+
void OnStart()
{
    //--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    //--- parameters of request
    request.action =TRADE_ACTION_DEAL; // type of trade operation
    request.symbol =Symbol(); // symbol
    request.volume =0.1; // volume of 0.1 lot
    request.type =ORDER_TYPE_BUY; // order type
    request.price =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // price for opening
    request.deviation=5; // allowed deviation from
    request.magic =EXPERT_MAGIC; // MagicNumber of the order
    //--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
    //--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+
```

Example of the [TRADE_ACTION_DEAL](#) trade operation for opening a Sell position:


```

#define EXPERT_MAGIC 123456    // MagicNumber of the expert
//+-----+
//| Opening Sell position |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- parameters of request
    request.action    =TRADE_ACTION_DEAL;           // type of trade operation
    request.symbol    =Symbol();                   // symbol
    request.volume    =0.2;                         // volume of 0.2 lot
    request.type      =ORDER_TYPE_SELL;            // order type
    request.price      =SymbolInfoDouble(Symbol(),SYMBOL_BID); // price for opening
    request.deviation=5;                             // allowed deviation from
    request.magic      =EXPERT_MAGIC;               // MagicNumber of the order
//--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the
//--- information about the operation
    PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+

```

Example of the `TRADE_ACTION_DEAL` trade operation for closing positions:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Closing all positions |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
for(int i=total-1; i>=0; i--)
{
//--- parameters of the order
ulong position_ticket=PositionGetTicket(i);
string position_symbol=PositionGetString(POSITION_SYMBOL);
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
ulong magic=PositionGetInteger(POSITION_MAGIC);
double volume=PositionGetDouble(POSITION_VOLUME);
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- output information about the position
PrintFormat("#%I64u %s %s %.2f %s [%I64d]",
            position_ticket,
            position_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            magic);
//--- if the MagicNumber matches
if(magic==EXPERT_MAGIC)
{
//--- zeroing the request and result values
ZeroMemory(request);
ZeroMemory(result);
//--- setting the operation parameters
request.action =TRADE_ACTION_DEAL; // type of trade operation
request.position =position_ticket; // ticket of the position
request.symbol =position_symbol; // symbol
request.volume =volume; // volume of the position
request.deviation=5; // allowed deviation from the price
request.magic =EXPERT_MAGIC; // MagicNumber of the position
//--- set the price and order type depending on the position type
if(type==POSITION_TYPE_BUY)
{
request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
request.type =ORDER_TYPE_SELL;
}
else
{
request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
request.type =ORDER_TYPE_BUY;
}
//--- output information about the closure
PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToString(type));
//--- send the request
if(!OrderSend(request,result))
PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
//---
}
}

```



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

Example of the [TRADE_ACTION_PENDING](#) trade operation for placing a pending order:


```

#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456 // MagicNumber of the expert
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT; // order type
//+-----+
//| Placing pending orders |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request={0};
MqlTradeResult result={0};
//--- parameters to place a pending order
request.action =TRADE_ACTION_PENDING; // type of trade operation
request.symbol =Symbol(); // symbol
request.volume =0.1; // volume of 0.1 lots
request.deviation=2; // allowed deviation in points
request.magic =EXPERT_MAGIC; // MagicNumber of the expert
int offset = 50; // offset from the order trigger price
double price; // order trigger price
double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT); // value of point in pips
int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS); // number of digits in price
//--- checking the type of operation
if(orderType==ORDER_TYPE_BUY_LIMIT)
{
request.type =ORDER_TYPE_BUY_LIMIT; // order type
price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // price for opening order
request.price =NormalizeDouble(price,digits); // normalized price
}
else if(orderType==ORDER_TYPE_SELL_LIMIT)
{
request.type =ORDER_TYPE_SELL_LIMIT; // order type
price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for opening order
request.price =NormalizeDouble(price,digits); // normalized price
}
else if(orderType==ORDER_TYPE_BUY_STOP)
{
request.type =ORDER_TYPE_BUY_STOP; // order type
price =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for opening order
request.price=NormalizeDouble(price,digits); // normalized price
}
else if(orderType==ORDER_TYPE_SELL_STOP)
{
request.type =ORDER_TYPE_SELL_STOP; // order type
price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // price for opening order
request.price =NormalizeDouble(price,digits); // normalized price
}
else Alert("This example is only for placing pending orders"); // if not pending
//--- send the request
if(!OrderSend(request,result))
PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+

```

Example of the **TRADE_ACTION_SLTP** trade operation for modifying the Stop Loss and Take Profit values of an open position:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Modification of Stop Loss and Take Profit of position |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
for(int i=0; i<total; i++)
{
//--- parameters of the order
ulong position_ticket=PositionGetTicket(i); // ticket of the position
string position_symbol=PositionGetString(POSITION_SYMBOL); // symbol
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // number of digits
ulong magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber of the position
double volume=PositionGetDouble(POSITION_VOLUME); // volume of the position
double sl=PositionGetDouble(POSITION_SL); // Stop Loss of the position
double tp=PositionGetDouble(POSITION_TP); // Take Profit of the position
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- output information about the position
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
            position_ticket,
            position_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{

```



```

//--- calculate the current price levels
double price=PositionGetDouble(POSITION_PRICE_OPEN);
double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
int stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_LEVEL);
double price_level;
//--- if the minimum allowed offset distance in points from the current close
if(stop_level<=0)
    stop_level=150; // set the offset distance of 150 points from the current close
else
    stop_level+=50; // set the offset distance to (SYMBOL_TRADE_STOPS_LEVEL + 50)

//--- calculation and rounding of the Stop Loss and Take Profit values
price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
if(type==POSITION_TYPE_BUY)
{
    sl=NormalizeDouble(bid-price_level,digits);
    tp=NormalizeDouble(ask+price_level,digits);
}
else
{
    sl=NormalizeDouble(ask+price_level,digits);
    tp=NormalizeDouble(bid-price_level,digits);
}
//--- zeroing the request and result values
ZeroMemory(request);
ZeroMemory(result);
//--- setting the operation parameters
request.action =TRADE_ACTION_SLTP; // type of trade operation
request.position=position_ticket; // ticket of the position
request.symbol=position_symbol; // symbol
request.sl =sl; // Stop Loss of the position
request.tp =tp; // Take Profit of the position
request.magic=EXPERT_MAGIC; // MagicNumber of the position
//--- output information about the modification
PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToString(TRADE_ACTION_SLTP));
//--- send the request
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
}
}
//+-----+

```

Example of the `TRADE_ACTION_MODIFY` trade operation for modifying the price levels of pending orders:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Modification of pending orders |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request={0};
MqlTradeResult result={0};
int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
for(int i=0; i<total; i++)
{
//--- parameters of the order
ulong order_ticket=OrderGetTicket(i); // order ticket
string order_symbol=Symbol(); // symbol
int digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS); // number of
ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber
double volume=OrderGetDouble(ORDER_VOLUME_CURRENT); // current volume
double sl=OrderGetDouble(ORDER_SL); // current Stop Loss
double tp=OrderGetDouble(ORDER_TP); // current Take Profit
ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // type of the order
int offset = 50; // offset from the order price
double price; // order trigger price
double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT); // value of the point
//--- output information about the order
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
            order_ticket,
            order_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{
    request.action=TRADE_ACTION_MODIFY; // type of trade request
    request.order = OrderGetTicket(i); // order ticket
    request.symbol =Symbol(); // symbol
    request.deviation=5; // allowed deviation
//--- setting the price level, Take Profit and Stop Loss of the order depending on the order type
    if(type==ORDER_TYPE_BUY_LIMIT)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
        request.tp = NormalizeDouble(price+offset*point,digits);
        request.sl = NormalizeDouble(price-offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(type==ORDER_TYPE_SELL_LIMIT)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
        request.tp = NormalizeDouble(price-offset*point,digits);
        request.sl = NormalizeDouble(price+offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(type==ORDER_TYPE_BUY_STOP)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
        request.tp = NormalizeDouble(price+offset*point,digits);
    }
}
}

```



```

        request.sl = NormalizeDouble(price-offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normal
    }
    else if(type==ORDER_TYPE_SELL_STOP)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
        request.tp = NormalizeDouble(price-offset*point,digits);
        request.sl = NormalizeDouble(price+offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normal
    }
    //--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send th
    //--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
    //--- zeroing the request and result values
    ZeroMemory(request);
    ZeroMemory(result);
}
}
}
//+-----+

```

Example of the **TRADE_ACTION_REMOVE** trade operation for deleting pending orders:

```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Deleting pending orders |
//+-----+
void OnStart()
{
    //--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    int total=OrdersTotal(); // total number of placed pending orders
    //--- iterate over all placed pending orders
    for(int i=total-1; i>=0; i--)
    {
        ulong order_ticket=OrderGetTicket(i); // order ticket
        ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber of the c
        //--- if the MagicNumber matches
        if(magic==EXPERT_MAGIC)
        {
            //--- zeroing the request and result values
            ZeroMemory(request);
            ZeroMemory(result);
            //--- setting the operation parameters
            request.action=TRADE_ACTION_REMOVE; // type of trade operat
            request.order = order_ticket; // order ticket
            //--- send the request
            if(!OrderSend(request,result))
                PrintFormat("OrderSend error %d",GetLastError()); // if unable to send th
            //--- information about the operation
            PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
        }
    }
}
//+-----+

```


Example of the [TRADE_ACTION_CLOSE_BY](#) trade operation for closing positions by opposite positions:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Close all positions by opposite positions |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
for(int i=total-1; i>=0; i--)
{
//--- parameters of the order
ulong position_ticket=PositionGetTicket(i);
string position_symbol=PositionGetString(POSITION_SYMBOL);
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
ulong magic=PositionGetInteger(POSITION_MAGIC);
double volume=PositionGetDouble(POSITION_VOLUME);
double sl=PositionGetDouble(POSITION_SL);
double tp=PositionGetDouble(POSITION_TP);
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- output information about the position
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
            position_ticket,
            position_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- if the MagicNumber matches
if(magic==EXPERT_MAGIC)
{
    for(int j=0; j<i; j++)
    {
        string symbol=PositionGetSymbol(j); // symbol of the opposite position
        //--- if the symbols of the opposite and initial positions match
        if(symbol==position_symbol && PositionGetInteger(POSITION_MAGIC)==EXPERT_M
        {
            //--- set the type of the opposite position
            ENUM_POSITION_TYPE type_by=(ENUM_POSITION_TYPE)PositionGetInteger(POSITI
            //--- leave, if the types of the initial and opposite positions match
            if(type==type_by)
                continue;
            //--- zeroing the request and result values
            ZeroMemory(request);
            ZeroMemory(result);
            //--- setting the operation parameters
            request.action=TRADE_ACTION_CLOSE_BY; // type c
            request.position=position_ticket; // ticket
            request.position_by=PositionGetInteger(POSITION_TICKET); // ticket
            //request.symbol =position_symbol;
            request.magic=EXPERT_MAGIC; // Magic
            //--- output information about the closure by opposite position
            PrintFormat("Close #%I64d %s %s by #%I64d",position_ticket,position_syn
            //--- send the request
            if(!OrderSend(request,result))
                PrintFormat("OrderSend error %d",GetLastError()); // if unable to se

```



```
        //--- information about the operation
        PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result.order);
    }
}
}
}
//+-----+
```


Trade Transaction Types

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using [OrderSend](#) and [OrderSendAsync](#) functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions.

To let a programmer to track the actions performed in relation to a trade account, [OnTradeTransaction](#) function has been provided. This handler allows to get trade transactions applied to an account in MQL5 application. Trade transaction description is submitted in [OnTradeTransaction](#) first parameter using [MqlTradeTransaction](#) structure.

Trade transaction type is submitted in the type parameter of [MqlTradeTransaction](#) structure. Possible types of trade transactions are described by the following enumeration:

ENUM_TRADE_TRANSACTION_TYPE

Identifier	Description
TRADE_TRANSACTION_ORDER_ADD	Adding a new open order.
TRADE_TRANSACTION_ORDER_UPDATE	Updating an open order. The updates include not only evident changes from the client terminal or a trade server sides but also changes of an order state when setting it (for example, transition from ORDER_STATE_STARTED to ORDER_STATE_PLACED or from ORDER_STATE_PLACED to ORDER_STATE_PARTIAL , etc.).
TRADE_TRANSACTION_ORDER_DELETE	Removing an order from the list of the open ones. An order can be deleted from the open ones as a result of setting an appropriate request or execution (filling) and moving to the history.

TRADE_TRANSACTION_DEAL_ADD	Adding a deal to the history. The action is performed as a result of an order execution or performing operations with an account balance.
TRADE_TRANSACTION_DEAL_UPDATE	Updating a deal in the history. There may be cases when a previously executed deal is changed on a server. For example, a deal has been changed in an external trading system (exchange) where it was previously transferred by a broker.
TRADE_TRANSACTION_DEAL_DELETE	Deleting a deal from the history. There may be cases when a previously executed deal is deleted from a server. For example, a deal has been deleted in an external trading system (exchange) where it was previously transferred by a broker.
TRADE_TRANSACTION_HISTORY_ADD	Adding an order to the history as a result of execution or cancellation.
TRADE_TRANSACTION_HISTORY_UPDATE	Changing an order located in the orders history. This type is provided for enhancing functionality on a trade server side.
TRADE_TRANSACTION_HISTORY_DELETE	Deleting an order from the orders history. This type is provided for enhancing functionality on a trade server side.
TRADE_TRANSACTION_POSITION	Changing a position not related to a deal execution. This type of transaction shows that a position has been changed on a trade server side. Position volume, open price, Stop Loss and Take Profit levels can be changed. Data on changes are submitted in MqlTradeTransaction structure via <code>OnTradeTransaction</code> handler. Position change (adding, changing or closing), as a result of a deal execution, does not lead to the occurrence of <code>TRADE_TRANSACTION_POSITION</code> transaction.
TRADE_TRANSACTION_REQUEST	Notification of the fact that a trade request has been processed by a server and processing result has been received. Only type field (trade transaction type) must be analyzed for such transactions in MqlTradeTransaction structure. The second and third parameters of OnTradeTransaction (request and result) must be analyzed for additional data.

Depending on a trade transaction type, various parameters are filled in `MqlTradeTransaction` structure describing it. A detailed description of submitted data is shown in "[Structure of a Trade Transaction](#)".

See also

Structure of a Trade Transaction, [OnTradeTransaction](#)

Trade Orders in Depth Of Market

For equity securities, the Depth of Market window is available, where you can see the current Buy and Sell orders. Desired direction of a trade operation, required amount and requested price are specified for each order.

To obtain information about the current state of the DOM by MQL5 means, the [MarketBookGet\(\)](#) function is used, which places the DOM "screen shot" into the [MqlBookInfo](#) array of structures. Each element of the array in the *type* field contains information about the direction of the order - the value of the ENUM_BOOK_TYPE enumeration.

ENUM_BOOK_TYPE

Identifier	Description
BOOK_TYPE_SELL	Sell order (Offer)
BOOK_TYPE_BUY	Buy order (Bid)
BOOK_TYPE_SELL_MARKET	Sell order by Market
BOOK_TYPE_BUY_MARKET	Buy order by Market

See also

[Structures and classes](#), [Structure of the DOM](#), [Trade operation types](#), [Market Info](#)

Signal Properties

The following enumerations are used when working with trading signals and signal copy settings.

Enumeration of [double](#) type properties of the trading signal:

ENUM_SIGNAL_BASE_DOUBLE

ID	Description
SIGNAL_BASE_BALANCE	Account balance
SIGNAL_BASE_EQUITY	Account equity
SIGNAL_BASE_GAIN	Account gain
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown
SIGNAL_BASE_PRICE	Signal subscription price
SIGNAL_BASE_ROI	Return on Investment (%)

Enumeration of [integer](#) type properties of the trading signal:

ENUM_SIGNAL_BASE_INTEGER

ID	Description
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)
SIGNAL_BASE_DATE_STARTED	Monitoring starting date
SIGNAL_BASE_ID	Signal ID
SIGNAL_BASE_LEVERAGE	Account leverage
SIGNAL_BASE_PIPS	Profit in pips
SIGNAL_BASE_RATING	Position in rating
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers
SIGNAL_BASE_TRADES	Number of trades
SIGNAL_BASE_TRADE_MODE	Account type (0-real, 1-demo, 2-contest)

Enumeration of [string](#) type properties of the trading signal:

ENUM_SIGNAL_BASE_STRING

ID	Description
SIGNAL_BASE_AUTHOR_LOGIN	Author login
SIGNAL_BASE_BROKER	Broker name (company)
SIGNAL_BASE_BROKER_SERVER	Broker server

SIGNAL_BASE_NAME	Signal name
SIGNAL_BASE_CURRENCY	Signal base currency

Enumeration of [double](#) type properties of the signal copy settings:

ENUM_SIGNAL_INFO_DOUBLE

ID	Description
SIGNAL_INFO_EQUITY_LIMIT	Equity limit
SIGNAL_INFO_SLIPPAGE	Slippage (used when placing market orders in synchronization of positions and copying of trades)
SIGNAL_INFO_VOLUME_PERCENT	Maximum percent of deposit used (%), r/o

Enumeration of [integer](#) type properties of the signal copy settings:

ENUM_SIGNAL_INFO_INTEGER

ID	Description
SIGNAL_INFO_CONFIRMATIONS_DISABLED	The flag enables synchronization without confirmation dialog
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag
SIGNAL_INFO_DEPOSIT_PERCENT	Deposit percent (%)
SIGNAL_INFO_ID	Signal id, r/o
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o

Enumeration of [string](#) type properties of the signal copy settings:

ENUM_SIGNAL_INFO_STRING

ID	Description
SIGNAL_INFO_NAME	Signal name, r/o

See also

[Trade signals](#)

Named Constants

All constants used in MQL5 can be divided into the following groups:

- [Predefined macro substitutions](#) - values are substituted during compilation;
- [Mathematical constants](#) - values of some mathematical expressions;
- [Numerical type constants](#) - some of the simple type restrictions;
- [Uninitialization reason codes](#) - description of uninitialization reasons;
- [Checking Object Pointer](#) - enumeration of types of pointers returned by the [CheckPointer\(\)](#) function;
- [Other constants](#) - all other constants.

Predefined Macro Substitutions

To simplify the debugging process and obtain information about operation of a mql5-program, there are special macro constant, values of which are set at the moment of compilation. The easiest way to use these constants is outputting values by the [Print\(\)](#) function, as it's shown in the example.

Constant	Description
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)
__DATETIME__	File compilation date and time
__LINE__	Line number in the source code, in which the macro is located
__FILE__	Name of the currently compiled file
__PATH__	An absolute path to the file that is currently being compiled
__FUNCTION__	Name of the function, in whose body the macro is located
__FUNCSIG__	Signature of the function in whose body the macro is located. Logging of the full description of functions can be useful in the identification of overloaded functions
__MQLBUILD__, __MQL5BUILD__	Compiler build number

Example:

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"

//+-----+
//| Expert initialization function |
//+-----+
void OnInit()
{
    //--- an example of information output at Expert Advisor initialization
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__ );
    //--- set the interval between the timer events
    EventSetTimer(5);
    //---
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- an example of information output at Expert Advisor deinitialization
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__ );
}
```



```

//---
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//--- information output at tick receipt
    Print(" __MQLBUILD__ = ", __MQLBUILD__, " __FILE__ = ", __FILE__ );
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__ );
    test1(__FUNCTION__);
    test2();
//---
}
//+-----+
//| test1 |
//+-----+
void test1(string par)
{
//--- information output inside the function
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__, " par = ", par);
}
//+-----+
//| test2 |
//+-----+
void test2()
{
//--- information output inside the function
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__ );
}
//+-----+
//| OnTimer event handler |
//+-----+
void OnTimer()
{
//---
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__ );
    test1(__FUNCTION__);
}

```


Mathematical Constants

Special constants containing values are reserved for some mathematical expressions. These constants can be used in any place of the program instead of calculating their values using [mathematical functions](#).

Constant	Description	Value
M_E	e	2.71828182845904523536
M_LOG2E	$\log_2(e)$	1.44269504088896340736
M_LOG10E	$\log_{10}(e)$	0.434294481903251827651
M_LN2	$\ln(2)$	0.693147180559945309417
M_LN10	$\ln(10)$	2.30258509299404568402
M_PI	pi	3.14159265358979323846
M_PI_2	$\pi/2$	1.57079632679489661923
M_PI_4	$\pi/4$	0.785398163397448309616
M_1_PI	$1/\pi$	0.318309886183790671538
M_2_PI	$2/\pi$	0.636619772367581343076
M_2_SQRTPI	$2/\sqrt{\pi}$	1.12837916709551257390
M_SQRT2	$\sqrt{2}$	1.41421356237309504880
M_SQRT1_2	$1/\sqrt{2}$	0.707106781186547524401

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- print the values of constants
    Print("M_E = ", DoubleToString(M_E, 16));
    Print("M_LOG2E = ", DoubleToString(M_LOG2E, 16));
    Print("M_LOG10E = ", DoubleToString(M_LOG10E, 16));
    Print("M_LN2 = ", DoubleToString(M_LN2, 16));
    Print("M_LN10 = ", DoubleToString(M_LN10, 16));
    Print("M_PI = ", DoubleToString(M_PI, 16));
    Print("M_PI_2 = ", DoubleToString(M_PI_2, 16));
    Print("M_PI_4 = ", DoubleToString(M_PI_4, 16));
    Print("M_1_PI = ", DoubleToString(M_1_PI, 16));
    Print("M_2_PI = ", DoubleToString(M_2_PI, 16));
    Print("M_2_SQRTPI = ", DoubleToString(M_2_SQRTPI, 16));
    Print("M_SQRT2 = ", DoubleToString(M_SQRT2, 16));
    Print("M_SQRT1_2 = ", DoubleToString(M_SQRT1_2, 16));
}
```



```
}
```


Numerical Type Constants

Each simple numerical type is intended for a certain type of tasks and allows optimizing the operation of a mql5-program when used correctly. For a better code readability and correct handling of calculation results, there are constants which allow to receive information about restrictions set to a certain type of simple data.

Constant	Description	Value
CHAR_MIN	Minimal value, which can be represented by char type	-128
CHAR_MAX	Maximal value, which can be represented by char type	127
UCHAR_MAX	Maximal value, which can be represented by uchar type	255
SHORT_MIN	Minimal value, which can be represented by short type	-32768
SHORT_MAX	Maximal value, which can be represented by short type	32767
USHORT_MAX	Maximal value, which can be represented by ushort type	65535
INT_MIN	Minimal value, which can be represented by int type	-2147483648
INT_MAX	Maximal value, which can be represented by int type	2147483647
UINT_MAX	Maximal value, which can be represented by uint type	4294967295
LONG_MIN	Minimal value, which can be represented by long type	-9223372036854775808
LONG_MAX	Maximal value, which can be represented by long type	9223372036854775807
ULONG_MAX	Maximal value, which can be represented by ulong type	18446744073709551615
DBL_MIN	Minimal positive value, which can be represented by double type	2.2250738585072014e-308
DBL_MAX	Maximal value, which can be represented by double type	1.7976931348623158e+308
DBL_EPSILON	Minimal value, which satisfies the condition: $1.0 + \text{DBL_EPSILON} \neq 1.0$ (for double type)	2.2204460492503131e-016

DBL_DIG	Number of significant decimal digits for double type	15
DBL_MANT_DIG	Number of bits in a mantissa for double type	53
DBL_MAX_10_EXP	Maximal decimal value of exponent degree for double type	308
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	1024
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	(-307)
DBL_MIN_EXP	Minimal binary value of exponent degree for double type	(-1021)
FLT_MIN	Minimal positive value, which can be represented by float type	1.175494351e-38
FLT_MAX	Maximal value, which can be represented by float type	3.402823466e+38
FLT_EPSILON	Minimal value, which satisfies the condition: $1.0 + \text{DBL_EPSILON} \neq 1.0$ (for float type)	1.192092896e-07
FLT_DIG	Number of significant decimal digits for float type	6
FLT_MANT_DIG	Number of bits in a mantissa for float type	24
FLT_MAX_10_EXP	Maximal decimal value of exponent degree for float type	38
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	128
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	-37
FLT_MIN_EXP	Minimal binary value of exponent degree for float type	(-125)

Example:

```
void OnStart()
{
    //--- print the constant values
```



```
printf("CHAR_MIN = %d",CHAR_MIN);
printf("CHAR_MAX = %d",CHAR_MAX);
printf("UCHAR_MAX = %d",UCHAR_MAX);
printf("SHORT_MIN = %d",SHORT_MIN);
printf("SHORT_MAX = %d",SHORT_MAX);
printf("USHORT_MAX = %d",USHORT_MAX);
printf("INT_MIN = %d",INT_MIN);
printf("INT_MAX = %d",INT_MAX);
printf("UINT_MAX = %u",UINT_MAX);
printf("LONG_MIN = %I64d",LONG_MIN);
printf("LONG_MAX = %I64d",LONG_MAX);
printf("ULONG_MAX = %I64u",ULONG_MAX);
printf("EMPTY_VALUE = %.16e",EMPTY_VALUE);
printf("DBL_MIN = %.16e",DBL_MIN);
printf("DBL_MAX = %.16e",DBL_MAX);
printf("DBL_EPSILON = %.16e",DBL_EPSILON);
printf("DBL_DIG = %d",DBL_DIG);
printf("DBL_MANT_DIG = %d",DBL_MANT_DIG);
printf("DBL_MAX_10_EXP = %d",DBL_MAX_10_EXP);
printf("DBL_MAX_EXP = %d",DBL_MAX_EXP);
printf("DBL_MIN_10_EXP = %d",DBL_MIN_10_EXP);
printf("DBL_MIN_EXP = %d",DBL_MIN_EXP);
printf("FLT_MIN = %.8e",FLT_MIN);
printf("FLT_MAX = %.8e",FLT_MAX);
printf("FLT_EPSILON = %.8e",FLT_EPSILON);
}
```


Uninitialization Reason Codes

Uninitialization reason [codes](#) are returned by the [UninitializeReason\(\)](#) function. The possible values are the following:

Constant	Value	Description
REASON_PROGRAM	0	Expert Advisor terminated its operation by calling the ExpertRemove() function
REASON_REMOVE	1	Program has been deleted from the chart
REASON_RECOMPILE	2	Program has been recompiled
REASON_CHARTCHANGE	3	Symbol or chart period has been changed
REASON_CHARTCLOSE	4	Chart has been closed
REASON_PARAMETERS	5	Input parameters have been changed by a user
REASON_ACCOUNT	6	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings
REASON_TEMPLATE	7	A new template has been applied
REASON_INITFAILED	8	This value means that OnInit() handler has returned a nonzero value
REASON_CLOSE	9	Terminal has been closed

The uninitialization reason code is also passed as a parameter of the predetermined function [OnDeinit](#)(const int reason).

Example:

```
//+-----+
//| get text description |
//+-----+
string getUninitReasonText(int reasonCode)
{
    string text="";
    //---
    switch(reasonCode)
    {
        case REASON_ACCOUNT:
            text="Account was changed";break;
    }
```



```

        case REASON_CHARTCHANGE:
            text="Symbol or timeframe was changed";break;
        case REASON_CHARTCLOSE:
            text="Chart was closed";break;
        case REASON_PARAMETERS:
            text="Input-parameter was changed";break;
        case REASON_RECOMPILE:
            text="Program "+__FILE__+" was recompiled";break;
        case REASON_REMOVE:
            text="Program "+__FILE__+" was removed from chart";break;
        case REASON_TEMPLATE:
            text="New template was applied to chart";break;
        default:text="Another reason";
    }
//---
    return text;
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- The first way to get the uninitialization reason code
    Print(__FUNCTION__,"_Uninitialization reason code = ",reason);
//--- The second way to get the uninitialization reason code
    Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason));
}

```


Checking Object Pointer

The [CheckPointer\(\)](#) function is used for checking the type of the [object pointer](#). The function returns a value of the ENUM_POINTER_TYPE enumeration. If an incorrect pointer is used, the program execution will be immediately terminated.

Objects created by the [new\(\)](#) operator are of POINTER_DYNAMIC type. The [delete\(\) operator](#) can and should be used only for such pointers.

All other pointers are of POINTER_AUTOMATIC type, which means that this object has been created automatically by the mql5 program environment. Such objects are deleted automatically after being used.

ENUM_POINTER_TYPE

Constant	Description
POINTER_INVALID	Incorrect pointer
POINTER_DYNAMIC	Pointer of the object created by the new() operator
POINTER_AUTOMATIC	Pointer of any objects created automatically (not using new())

See also

[Runtime errors](#), [Object Delete Operator delete](#), [CheckPointer](#)

Other Constants

The CLR_NONE constant is used to outline the absence of color, it means that the [graphical object](#) or [graphical series](#) of an indicator will not be plotted. This constant was not included into the [Web-color](#) constants list, but it can be applied everywhere where the color arguments are required.

The INVALID_HANDLE constant can be used for checking file handles (see [FileOpen\(\)](#) and [FileFindFirst\(\)](#)).

Constant	Description	Value
CHARTS_MAX	The maximum possible number of simultaneously open charts in the terminal	100
clrNONE	Absence of color	-1
EMPTY_VALUE	Empty value in an indicator buffer	DBL_MAX
INVALID_HANDLE	Incorrect handle	-1
IS_DEBUG_MODE	Flag that a mq5-program operates in debug mode	non zero in debug mode, otherwise zero
IS_PROFILE_MODE	Flag that a mq5-program operates in profiling mode	non zero in profiling mode, otherwise zero
NULL	Zero for any types	0
WHOLE_ARRAY	Means the number of items remaining until the end of the array, i.e., the entire array will be processed	-1
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	-1

The EMPTY_VALUE constant usually corresponds to the values of indicators that are not shown in the chart. For example, for built-in indicator Standard Deviation with a period of 20, the line for the first 19 bars in the history is not shown in the chart. If you create a handle of this indicator with the [iStdDev\(\)](#) function and copy it to an array of indicator values for these bars through [CopyBuffer\(\)](#), then these values will be equal to EMPTY_VALUE.

You can choose to specify for [a custom indicator](#) your own empty value of the indicator, when the indicator shouldn't be drawn in the chart. Use the [PlotIndexSetDouble\(\)](#) function with the [PLOT_EMPTY_VALUE](#) modifier.

The [NULL](#) constant can be assigned to a variable of any simple type or to an object structure or class pointer. The NULL assignment for a string variable means the full deinitialization of this variable.

The WRONG_VALUE constant is intended for cases, when it is necessary to return value of an [enumeration](#), and this must be a wrong value. For example, when we need to inform that a return value is a value from this enumeration. Let's consider as an example some function CheckLineStyle(), which returns the line style for an object, specified by its name. If at style check by

ObjectGetInteger() the result is true, a value from [ENUM_LINE_STYLE](#) is returned; otherwise WRONG_VALUE is returned.

```
void OnStart()
{
    if (CheckLineStyle("MyChartObject")==WRONG_VALUE)
        printf("Error line style getting.");
}

//+-----+
//| returns the line style for an object specified by its name |
//+-----+
ENUM_LINE_STYLE CheckLineStyle(string name)
{
    long style;
//---
    if (ObjectGetInteger(0,name,OBJPROP_STYLE,0,style))
        return((ENUM_LINE_STYLE)style);
    else
        return(WRONG_VALUE);
}
```

The WHOLE_ARRAY constant is intended for functions that require specifying the number of elements in processed arrays:

- [ArrayCopy\(\)](#);
- [ArrayMinimum\(\)](#);
- [ArrayMaximum\(\)](#);
- [FileReadArray\(\)](#);
- [FileWriteArray\(\)](#).

If you want to specify that all the array values from a specified position till the end must be processed, you should specify just the WHOLE_ARRAY value.

IS_PROFILE_MODE constant allows changing a program operation for correct data collection in the profiling mode. Profiling allows measuring the execution time of the individual program fragments (usually comprising functions), as well as calculating the number of such calls. Sleep() function calls can be disabled to determine the execution time in the profiling mode, like in this example:

```
//--- Sleep can greatly affect (change) profiling result
if(!IS_PROFILE_MODE) Sleep(100); // disabling Sleep() call in the profiling mode
```

IS_PROFILE_MODE constant value is set by the compiler during the compilation, while it is set to zero in conventional mode. When launching a program in the profiling mode, a special compilation is performed and IS_PROFILE_MODE is replaced with a non-zero value.

The IS_DEBUG_MODE constant can be useful when you need to slightly change the operation of a mql5 program in the debugging mode. For example, in debug mode you may need to display additional debugging information in the terminal log or create additional graphical objects in a chart.

The following example creates a Label object and sets its description and color depending on the script running mode. In order to run a script in the debug mode from MetaEditor, press F5. If you run the script from the browser window in the terminal, then the color and text of the object Label will be different.

Example:

```
//+-----+
//|                                     Check_DEBUG_MODE.mq5 |
//|                                     Copyright © 2009, MetaQuotes Software Corp. |
//|                                     https://www.metaquotes.net |
//+-----+
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
    string label_name="invisible_label";
    if(ObjectFind(0,label_name)<0)
    {
        Print("Object",label_name,"not found. Error code = ",GetLastError());
        //--- create Label
        ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
        //--- set X coordinate
        ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
        //--- set Y coordinate
        ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
        ResetLastError();
        if(IS_DEBUG_MODE) // debug mode
        {
            //--- show message about the script execution mode
            ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
            //--- set text color to red
            if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
                Print("Unable to set the color. Error",GetLastError());
        }
    }
    else // operation mode
    {
        ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
        //--- set text color to invisible
        if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
            Print("Unable to set the color. Error ",GetLastError());
    }
    ChartRedraw();
    DebugBreak(); // here termination will occur, if we are in debug mode
}
```



```
}
```

Crypt Methods

The ENUM_CRYPT_METHOD enumeration is used to specify the data transformation method, used in [CryptEncode\(\)](#) and [CryptDecode\(\)](#) functions.

ENUM_CRYPT_METHOD

Constant	Description
CRYPT_BASE64	BASE64
CRYPT_AES128	AES encryption with 128 bit key (16 bytes)
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)
CRYPT_DES	DES encryption with 56 bit key (7 bytes)
CRYPT_HASH_SHA1	SHA1 HASH calculation
CRYPT_HASH_SHA256	SHA256 HASH calculation
CRYPT_HASH_MD5	MD5 HASH calculation
CRYPT_ARCH_ZIP	ZIP archives

See also

[DebugBreak](#), [Executed MQL5 program properties](#), [CryptEncode\(\)](#), [CryptDecode\(\)](#)

Data Structures

MQL5 Language offers 8 predefined [structures](#):

- [MqlDateTime](#) is intended for working with [date and time](#);
- [MqlParam](#) can send input parameters when creating a handle of the indicator using the [IndicatorCreate\(\)](#) function;
- [MqlRates](#) is intended for manipulating the [historical data](#), it contains information about the price, volume and spread;
- [MqlBookInfo](#) is intended for obtaining information about the [Depth of Market](#);
- [MqlTradeRequest](#) is used for creating a trade request for [trade operations](#);
- [MqlTradeCheckResult](#) is intended for [checking](#) the prepared [trade request](#) before [sending](#) it;
- [MqlTradeResult](#) contains a trade server reply to a [trade request](#), sent by [OrderSend\(\)](#) function;
- [MqlTradeTransaction](#) contains description of a trade transaction;
- [MqlTick](#) is designed for fast retrieval of the most requested information about current prices.

MqlDateTime

The date type structure contains eight fields of the [int](#) type:

```
struct MqlDateTime
{
    int year;           // Year
    int mon;            // Month
    int day;            // Day
    int hour;           // Hour
    int min;            // Minutes
    int sec;            // Seconds
    int day_of_week;    // Day of week (0-Sunday, 1-Monday, ... ,6-Saturday)
    int day_of_year;    // Day number of the year (January 1st is assigned the number 1)
};
```

Note

The day number of the year `day_of_year` for the leap year, since March, will differ from a number of the corresponding day for a non-leap year.

Example:

```
void OnStart ()
{
    //---
    datetime date1=D'2008.03.01';
    datetime date2=D'2009.03.01';

    MqlDateTime str1,str2;
    TimeToStruct(date1,str1);
    TimeToStruct(date2,str2);
    printf("%02d.%02d.%4d, day of year = %d",str1.day,str1.mon,
        str1.year,str1.day_of_year);
    printf("%02d.%02d.%4d, day of year = %d",str2.day,str2.mon,
        str2.year,str2.day_of_year);
}
/* Result:
    01.03.2008, day of year = 60
    01.03.2009, day of year = 59
*/
```

See also

[TimeToStruct](#), [Structures and Classes](#)

The Structure of Input Parameters of Indicators (MqlParam)

The MqlParam structure has been specially designed to provide [input parameters](#) when creating the handle of [a technical indicator](#) using the [IndicatorCreate\(\)](#) function.

```
struct MqlParam
{
    ENUM_DATATYPE    type;           // type of the input parameter, value of
    long             integer_value;  // field to store an integer type
    double           double_value;   // field to store a double type
    string           string_value;    // field to store a string type
};
```

All input parameters of an indicator are transmitted in the form of an array of the MqlParam type, the *type* field of each element of this array specifies the type of data transmitted by the element. The indicator values must be first placed in the appropriate fields for each element (in *integer_value*, in *double_value* or *string_value*) depending on what value of [ENUM_DATATYPE](#) enumeration is specified in the *type* field.

If the IND_CUSTOM value is passed third as the indicator type to the [IndicatorCreate\(\)](#) function, the first element of the array of input parameters must have the *type* field with the value of TYPE_STRING from the [ENUM_DATATYPE](#) enumeration, and the *string_value* field must contain the name of the [custom indicator](#).

MqlRates

This structure stores information about the prices, volumes and spread.

```
struct MqlRates
{
    datetime time;           // Period start time
    double   open;           // Open price
    double   high;           // The highest price of the period
    double   low;            // The lowest price of the period
    double   close;          // Close price
    long     tick_volume;    // Tick volume
    int      spread;         // Spread
    long     real_volume;    // Trade volume
};
```

Example:

```
void OnStart()
{
    MqlRates rates[];
    int copied=CopyRates(NULL,0,0,100,rates);
    if(copied<=0)
        Print("Error copying price data ",GetLastError());
    else Print("Copied ",ArraySize(rates)," bars");
}
```

See also

[CopyRates](#), [Access to timeseries](#)

MqlBookInfo

It provides information about the market depth data.

```
struct MqlBookInfo
{
    ENUM_BOOK_TYPE    type;        // Order type from ENUM BOOK TYPE enumeration
    double            price;        // Price
    long              volume;       // Volume
};
```

Note

The MqlBookInfo structure is predefined, thus it doesn't require any declaration and description. To use the structure, just declare a variable of this type.

The DOM is available only for some symbols.

Example:

```
MqlBookInfo priceArray[];
bool getBook=MarketBookGet(NULL,priceArray);
if (getBook)
{
    int size=ArraySize(priceArray);
    Print("MarketBookInfo about ",Symbol());
}
else
{
    Print("Failed to receive DOM for the symbol ",Symbol());
}
```

See also

[MarketBookAdd](#), [MarketBookRelease](#), [MarketBookGet](#), [Trade Orders in DOM](#), [Data Types](#)

The Trade Request Structure (MqlTradeRequest)

Interaction between the client terminal and a trade server for executing the order placing operation is performed by using trade requests. The trade request is represented by the special predefined [structure](#) of MqlTradeRequest type, which contain all the fields necessary to perform trade deals. The request processing result is represented by the structure of [MqlTradeResult](#) type.

```
struct MqlTradeRequest
{
    ENUM_TRADE_REQUEST_ACTIONS    action;           // Trade operation type
    ulong                         magic;             // Expert Advisor ID (magic number)
    ulong                         order;             // Order ticket
    string                        symbol;            // Trade symbol
    double                        volume;            // Requested volume for a deal in lots
    double                        price;             // Price
    double                        stoplimit;         // StopLimit level of the order
    double                        sl;               // Stop Loss level of the order
    double                        tp;               // Take Profit level of the order
    ulong                         deviation;         // Maximal possible deviation from price
    ENUM_ORDER_TYPE              type;              // Order type
    ENUM_ORDER_TYPE_FILLING      type_filling;      // Order execution type
    ENUM_ORDER_TYPE_TIME         type_time;        // Order expiration type
    datetime                     expiration;        // Order expiration time (for the order)
    string                        comment;          // Order comment
    ulong                         position;          // Position ticket
    ulong                         position_by;       // The ticket of an opposite position
};
```

Fields description

Field	Description
action	Trade operation type. Can be one of the ENUM_TRADE_REQUEST_ACTIONS enumeration values.
magic	Expert Advisor ID. It allows organizing analytical processing of trade orders. Each Expert Advisor can set its own unique ID when sending a trade request.
order	Order ticket. It is used for modifying pending orders.
symbol	Symbol of the order. It is not necessary for order modification and position close operations.
volume	Requested order volume in lots. Note that the real volume of a deal will depend on the order execution type .

price	Price, reaching which the order must be executed. Market orders of symbols, whose execution type is "Market Execution" (SYMBOL_TRADE_EXECUTION_MARKET), of TRADE_ACTION_DEAL type, do not require specification of price.
stoplimit	The price value, at which the Limit pending order will be placed, when price reaches the <i>price</i> value (this condition is obligatory). Until then the pending order is not placed.
sl	Stop Loss price in case of the unfavorable price movement
tp	Take Profit price in the case of the favorable price movement
deviation	The maximal price deviation, specified in points
type	Order type. Can be one of the ENUM_ORDER_TYPE enumeration values.
type_filling	Order execution type. Can be one of the enumeration ENUM_ORDER_TYPE_FILLING values.
type_time	Order expiration type. Can be one of the enumeration ENUM_ORDER_TYPE_TIME values.
expiration	Order expiration time (for orders of ORDER_TIME_SPECIFIED type)
comment	Order comment
position	Ticket of a position. Should be filled in when a position is modified or closed to identify the position. As a rule it is equal to the ticket of the order, based on which the position was opened.
position_by	Ticket of an opposite position. Used when a position is closed by an opposite one open for the same symbol in the opposite direction.

When modifying or closing a position in the hedging system, make sure to specify its ticket (`MqlTradeRequest::position`). The ticket can also be specified in the netting system, though a position is identified by the symbol name.

For sending orders to perform [trade operations](#) it is necessary to use the [OrderSend\(\)](#) function. For each trade operation it is necessary to specify obligatory fields; optional fields also may be filled. There are seven possible cases to send a trade order:

Request Execution

This is a trade order to open a position in the Request Execution mode (trade upon requested prices). It requires to specify the following 9 fields:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Instant Execution

This is a trade order to open a position in the Instant Execution mode (trade by current prices). It requires specification of the following 9 fields:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Market Execution

This is a trade order to open a position in the Market Execution mode. It requires to specify the following 5 fields:

- action
- symbol
- volume
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Exchange Execution

This is a trade order to open a position in the Exchange Execution mode. It requires to specify the following 5 fields:

- action
- symbol
- volume

- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Example of the [TRADE_ACTION_DEAL](#) trade operation for opening a Buy position:

```
#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Opening Buy position |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- parameters of request
    request.action =TRADE_ACTION_DEAL; // type of trade operation
    request.symbol =Symbol(); // symbol
    request.volume =0.1; // volume of 0.1 lot
    request.type =ORDER_TYPE_BUY; // order type
    request.price =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // price for opening
    request.deviation=5; // allowed deviation from
    request.magic =EXPERT_MAGIC; // MagicNumber of the order
//--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+
```

Example of the [TRADE_ACTION_DEAL](#) trade operation for opening a Sell position:

```
#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Opening Sell position |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- parameters of request
    request.action =TRADE_ACTION_DEAL; // type of trade operation
    request.symbol =Symbol(); // symbol
    request.volume =0.2; // volume of 0.2 lot
    request.type =ORDER_TYPE_SELL; // order type
    request.price =SymbolInfoDouble(Symbol(),SYMBOL_BID); // price for opening
    request.deviation=5; // allowed deviation from
    request.magic =EXPERT_MAGIC; // MagicNumber of the order
//--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+
```


Example of the `TRADE_ACTION_DEAL` trade operation for closing positions:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Closing all positions |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
for(int i=total-1; i>=0; i--)
{
//--- parameters of the order
ulong position_ticket=PositionGetTicket(i);
string position_symbol=PositionGetString(POSITION_SYMBOL);
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
ulong magic=PositionGetInteger(POSITION_MAGIC);
double volume=PositionGetDouble(POSITION_VOLUME);
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- output information about the position
PrintFormat("#%I64u %s %s %.2f %s [%I64d]",
            position_ticket,
            position_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            magic);
//--- if the MagicNumber matches
if(magic==EXPERT_MAGIC)
{
//--- zeroing the request and result values
ZeroMemory(request);
ZeroMemory(result);
//--- setting the operation parameters
request.action =TRADE_ACTION_DEAL; // type of trade operation
request.position =position_ticket; // ticket of the position
request.symbol =position_symbol; // symbol
request.volume =volume; // volume of the position
request.deviation=5; // allowed deviation from the price
request.magic =EXPERT_MAGIC; // MagicNumber of the position
//--- set the price and order type depending on the position type
if(type==POSITION_TYPE_BUY)
{
request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
request.type =ORDER_TYPE_SELL;
}
else
{
request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
request.type =ORDER_TYPE_BUY;
}
//--- output information about the closure
PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToString(type));
//--- send the request
if(!OrderSend(request,result))
PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
//---
}
}

```



```

    }
}
//+-----+

```

SL & TP Modification

Trade order to modify the StopLoss and/or TakeProfit price levels. It requires to specify the following 4 fields:

- action
- symbol
- sl
- tp
- position

Example of the [TRADE_ACTION_SLTP](#) trade operation for modifying the Stop Loss and Take Profit values of an open position:

```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Modification of Stop Loss and Take Profit of position |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
for(int i=0; i<total; i++)
{
//--- parameters of the order
ulong position_ticket=PositionGetTicket(i); // ticket of the position
string position_symbol=PositionGetString(POSITION_SYMBOL); // symbol
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // number of digits
ulong magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber of the position
double volume=PositionGetDouble(POSITION_VOLUME); // volume of the position
double sl=PositionGetDouble(POSITION_SL); // Stop Loss of the position
double tp=PositionGetDouble(POSITION_TP); // Take Profit of the position
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- output information about the position
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
            position_ticket,
            position_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{

```



```

//--- calculate the current price levels
double price=PositionGetDouble(POSITION_PRICE_OPEN);
double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
int stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_LEVEL);
double price_level;
//--- if the minimum allowed offset distance in points from the current close
if(stop_level<=0)
    stop_level=150; // set the offset distance of 150 points from the current close
else
    stop_level+=50; // set the offset distance to (SYMBOL_TRADE_STOPS_LEVEL + 50)

//--- calculation and rounding of the Stop Loss and Take Profit values
price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
if(type==POSITION_TYPE_BUY)
{
    sl=NormalizeDouble(bid-price_level,digits);
    tp=NormalizeDouble(ask+price_level,digits);
}
else
{
    sl=NormalizeDouble(ask+price_level,digits);
    tp=NormalizeDouble(bid-price_level,digits);
}
//--- zeroing the request and result values
ZeroMemory(request);
ZeroMemory(result);
//--- setting the operation parameters
request.action =TRADE_ACTION_SLTP; // type of trade operation
request.position=position_ticket; // ticket of the position
request.symbol=position_symbol; // symbol
request.sl =sl; // Stop Loss of the position
request.tp =tp; // Take Profit of the position
request.magic=EXPERT_MAGIC; // MagicNumber of the position
//--- output information about the modification
PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToString(POSITION_TYPE_BUY));
//--- send the request
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
//--- information about the operation
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,position_ticket);
}
}
}
//+-----+

```

Pending Order

Trade order to place a pending order. It requires to specify the following 11 fields:

- action
- symbol
- volume
- price
- stoplimit
- sl
- tp
- type

- type_filling
- type_time
- expiration

Also it is possible to specify the "magic" and "comment" field values.

Example of the [TRADE_ACTION_PENDING](#) trade operation for placing a pending order:


```

#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456 // MagicNumber of the expert
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT; // order type
//+-----+
//| Placing pending orders |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- parameters to place a pending order
    request.action =TRADE_ACTION_PENDING; // type of trade request
    request.symbol =Symbol(); // symbol
    request.volume =0.1; // volume of 0.1 lots
    request.deviation=2; // allowed deviation
    request.magic =EXPERT_MAGIC; // MagicNumber
    int offset = 50; // offset from the price
    double price; // order trigger price
    double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT); // value of point
    int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS); // number of digits
//--- checking the type of operation
    if(orderType==ORDER_TYPE_BUY_LIMIT)
    {
        request.type =ORDER_TYPE_BUY_LIMIT; // order type
        price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // price for order
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(orderType==ORDER_TYPE_SELL_LIMIT)
    {
        request.type =ORDER_TYPE_SELL_LIMIT; // order type
        price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for order
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(orderType==ORDER_TYPE_BUY_STOP)
    {
        request.type =ORDER_TYPE_BUY_STOP; // order type
        price =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for order
        request.price=NormalizeDouble(price,digits); // normalized price
    }
    else if(orderType==ORDER_TYPE_SELL_STOP)
    {
        request.type =ORDER_TYPE_SELL_STOP; // order type
        price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // price for order
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else Alert("This example is only for placing pending orders"); // if not pending
//--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send
//--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+

```

Modify Pending Order

Trade order to modify the prices of a pending order. It requires to specify the following 7 fields:

- action

- order
- price
- sl
- tp
- type_time
- expiration

Example of the [TRADE_ACTION_MODIFY](#) trade operation for modifying the price levels of pending orders:


```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert
//+-----+
//| Modification of pending orders |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request={0};
MqlTradeResult result={0};
int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
for(int i=0; i<total; i++)
{
//--- parameters of the order
ulong order_ticket=OrderGetTicket(i); // order ticket
string order_symbol=Symbol(); // symbol
int digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS); // number of
ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber
double volume=OrderGetDouble(ORDER_VOLUME_CURRENT); // current volume
double sl=OrderGetDouble(ORDER_SL); // current Stop Loss
double tp=OrderGetDouble(ORDER_TP); // current Take Profit
ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // type of the order
int offset = 50; // offset from the order price
double price; // order trigger price
double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT); // value of the point
//--- output information about the order
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
            order_ticket,
            order_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{
    request.action=TRADE_ACTION_MODIFY; // type of trade request
    request.order = OrderGetTicket(i); // order ticket
    request.symbol =Symbol(); // symbol
    request.deviation=5; // allowed deviation
//--- setting the price level, Take Profit and Stop Loss of the order depending on the order type
    if(type==ORDER_TYPE_BUY_LIMIT)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
        request.tp = NormalizeDouble(price+offset*point,digits);
        request.sl = NormalizeDouble(price-offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(type==ORDER_TYPE_SELL_LIMIT)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
        request.tp = NormalizeDouble(price-offset*point,digits);
        request.sl = NormalizeDouble(price+offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normalized price
    }
    else if(type==ORDER_TYPE_BUY_STOP)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
        request.tp = NormalizeDouble(price+offset*point,digits);
    }
}
}

```



```

        request.sl = NormalizeDouble(price-offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normal
    }
    else if(type==ORDER_TYPE_SELL_STOP)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
        request.tp = NormalizeDouble(price-offset*point,digits);
        request.sl = NormalizeDouble(price+offset*point,digits);
        request.price =NormalizeDouble(price,digits); // normal
    }
    //--- send the request
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // if unable to send th
    //--- information about the operation
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
    //--- zeroing the request and result values
    ZeroMemory(request);
    ZeroMemory(result);
    }
}
}
//+-----+

```

Delete Pending Order

Trade order to delete a pending order. It requires to specify the following 2 fields:

- action
- order

Example of the [TRADE_ACTION_REMOVE](#) trade operation for deleting pending orders:

```

#define EXPERT_MAGIC 123456 // MagicNumber of the expert

```



```

//+-----+
//| Deleting pending orders |
//+-----+
void OnStart()
{
//--- declare and initialize the trade request and result of trade request
MqlTradeRequest request={0};
MqlTradeResult result={0};
int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
for(int i=total-1; i>=0; i--)
{
    ulong order_ticket=OrderGetTicket(i); // order ticket
    ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber of the order
    //--- if the MagicNumber matches
    if(magic==EXPERT_MAGIC)
    {
        //--- zeroing the request and result values
        ZeroMemory(request);
        ZeroMemory(result);
        //--- setting the operation parameters
        request.action=TRADE_ACTION_REMOVE; // type of trade operation
        request.order = order_ticket; // order ticket
        //--- send the request
        if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError()); // if unable to send the request
        //--- information about the operation
        PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,order_ticket);
    }
}
}
//+-----+

```

See also

[Structures and Classes](#), [Trade Functions](#), [Order Properties](#)

The Structure of Results of a Trade Request Check (MqlTradeCheckResult)

Before [sending](#) a [request](#) for a [trade operation](#) to a trade server, it is recommended to check it. The check is performed using the [OrderCheck\(\)](#) function, to which the checked request and a variable of the MqlTradeCheckResult structure type are passed. The check result will be written to this variable.

```
struct MqlTradeCheckResult
{
    uint      retcode;           // Reply code
    double     balance;          // Balance after the execution of the deal
    double     equity;           // Equity after the execution of the deal
    double     profit;           // Floating profit
    double     margin;           // Margin requirements
    double     margin_free;      // Free margin
    double     margin_level;     // Margin level
    string     comment;          // Comment to the reply code (description of the
};
```

Description of Fields

Field	Description
retcode	Return code
balance	Balance value that will be after the execution of the trade operation
equity	Equity value that will be after the execution of the trade operation
profit	Value of the floating profit that will be after the execution of the trade operation
margin	Margin required for the trade operation
margin_free	Free margin that will be left after the execution of the trade operation
margin_level	Margin level that will be set after the execution of the trade operation
comment	Comment to the reply code, error description

See also

[Trade Request Structure](#), [Structure for Current Prices](#), [OrderSend](#), [OrderCheck](#)

The Structure of a Trade Request Result (MqlTradeResult)

As result of a [trade request](#), a trade server returns data about the trade request processing result as a special predefined structure of MqlTradeResult type.

```
struct MqlTradeResult
{
    uint      retcode;           // Operation return code
    ulong     deal;             // Deal ticket, if it is performed
    ulong     order;            // Order ticket, if it is placed
    double    volume;           // Deal volume, confirmed by broker
    double    price;            // Deal price, confirmed by broker
    double    bid;              // Current Bid price
    double    ask;              // Current Ask price
    string     comment;          // Broker comment to operation (by default it is filled
    uint       request_id;       // Request ID set by the terminal during the dispatch
    uint       retcode_external; // Return code of an external trading system
};
```

Fields description

Field	Description
retcode	Return code of a trade server
deal	Deal ticket, if a deal has been performed. It is available for a trade operation of TRADE_ACTION_DEAL type
order	Order ticket, if a ticket has been placed. It is available for a trade operation of TRADE_ACTION_PENDING type
volume	Deal volume, confirmed by broker. It depends on the order filling type
price	Deal price, confirmed by broker. It depends on the <i>deviation</i> field of the trade request and/or on the trade operation
bid	The current market Bid price (request price)
ask	The current market Ask price (request price)
comment	The broker comment to operation (by default it is filled by description of trade server return code)
request_id	Request ID set by the terminal when sending to the trade server
retcode_external	The code of the error returned by an external trading system. The use and types of these

errors depend on the broker and the external trading system, to which trading operations are sent.
--

The trade operation result is returned to a variable of the `MqlTradeResult` type, which is passed as the second parameter to [OrderSend\(\)](#) to perform [trade operations](#).

The terminal fixes [request](#) ID in `request_id` field when sending it to the trade server using [OrdersSend\(\)](#) and [OrderSendAsync\(\)](#) functions. The terminal receives messages about performed transactions from the trade server and submits them for processing by [OnTradeTransaction\(\)](#) function containing the following components as parameters:

- description of the trade transaction in [MqlTradeTransaction](#) structure;
- description of the [trade request](#) sent from [OrderSend\(\)](#) or [OrdersSendAsync\(\)](#) function. Request ID is sent by the terminal to the trade server, while the request itself and its `request_id` are stored in the terminal memory;
- the trade request execution result as `MqlTradeResult` structure with `request_id` field containing ID of this request.

[OnTradeTransaction\(\)](#) function receives three input parameters but the last two should be analyzed only for transactions having [TRADE_TRANSACTION_REQUEST](#) type. In all other cases, data on the trade request and its execution result are not filled. Example of parameters analysis can be found at [Structure of a Trade Request](#).

Setting `request_id` by the terminal for the trade request when sending it to the server is mainly introduced for working with [OrderSendAsync\(\)](#) asynchronous function. This identifier allows to associate the performed action ([OrderSend](#) or [OrderSendAsync](#) functions call) with the result of this action sent to [OnTradeTransaction\(\)](#).

Example:

```
//+-----+
//| Sending a trade request with the result processing |
//+-----+
bool MyOrderSend(MqlTradeRequest request,MqlTradeResult result)
{
    //-- reset the last error code to zero
    ResetLastError();
    //-- send request
    bool success=OrderSend(request,result);
    //-- if the result fails - try to find out why
    if(!success)
    {
        int answer=result.retcode;
        Print("TradeLog: Trade request failed. Error = ",GetLastError());
        switch(answer)
        {
            //-- requote
            case 10004:
            {
```



```

Print("TRADE_RETCODE_REQUOTE");
Print("request.price = ",request.price,"    result.ask = ",
      result.ask," result.bid = ",result.bid);
break;
}
//--- order is not accepted by the server
case 10006:
{
    Print("TRADE_RETCODE_REJECT");
    Print("request.price = ",request.price,"    result.ask = ",
          result.ask," result.bid = ",result.bid);
    break;
}
//--- invalid price
case 10015:
{
    Print("TRADE_RETCODE_INVALID_PRICE");
    Print("request.price = ",request.price,"    result.ask = ",
          result.ask," result.bid = ",result.bid);
    break;
}
//--- invalid SL and/or TP
case 10016:
{
    Print("TRADE_RETCODE_INVALID_STOPS");
    Print("request.sl = ",request.sl," request.tp = ",request.tp);
    Print("result.ask = ",result.ask," result.bid = ",result.bid);
    break;
}
//--- invalid volume
case 10014:
{
    Print("TRADE_RETCODE_INVALID_VOLUME");
    Print("request.volume = ",request.volume,"    result.volume = ",
          result.volume);
    break;
}
//--- not enough money for a trade operation
case 10019:
{
    Print("TRADE_RETCODE_NO_MONEY");
    Print("request.volume = ",request.volume,"    result.volume = ",
          result.volume,"    result.comment = ",result.comment);
    break;
}
//--- some other reason, output the server response code
default:
{
    Print("Other answer = ",answer);

```



```
    }  
    }  
    //--- notify about the unsuccessful result of the trade request by returning false  
    return(false);  
    }  
    //--- OrderSend() returns true - repeat the answer  
    return(true);  
    }
```


Structure of a Trade Transaction (MqlTradeTransaction)

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using [OrderSend](#) and [OrderSendAsync](#) functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions.

Special [OnTradeTransaction\(\)](#) handler is provided in MQL5 to get trade transactions applied to an account. The first parameter of the handler gets [MqlTradeTransaction](#) structure describing [trade transactions](#).

```
struct MqlTradeTransaction
{
    ulong                deal;           // Deal ticket
    ulong                order;          // Order ticket
    string               symbol;         // Trade symbol name
    ENUM_TRADE_TRANSACTION_TYPE type;     // Trade transaction type
    ENUM_ORDER_TYPE      order_type;     // Order type
    ENUM_ORDER_STATE     order_state;    // Order state
    ENUM_DEAL_TYPE       deal_type;      // Deal type
    ENUM_ORDER_TYPE_TIME time_type;      // Order type by action period
    datetime             time_expiration; // Order expiration time
    double               price;          // Price
    double               price_trigger;   // Stop limit order activation price
    double               price_sl;       // Stop Loss level
    double               price_tp;       // Take Profit level
    double               volume;         // Volume in lots
    ulong                position;       // Position ticket
    ulong                position_by;    // Ticket of an opposite position
};
```

Fields Description

Field	Description
deal	Deal ticket.
order	Order ticket.
symbol	The name of the trading symbol, for which transaction is performed.
type	Trade transaction type. The value can be one of ENUM_TRADE_TRANSACTION_TYPE enumeration values.
order_type	Trade order type. The value can be one of ENUM_ORDER_TYPE enumeration values.
order_state	Trade order state. The value can be one of ENUM_ORDER_STATE enumeration values.
deal_type	Deal type. The value can be one of ENUM_DEAL_TYPE enumeration values.
type_time	Order type upon expiration. The value can be one of ENUM_ORDER_TYPE_TIME values.
time_expiration	Pending order expiration term (for orders of ORDER_TIME_SPECIFIED and ORDER_TIME_SPECIFIED_DAY types).
price	Price. Depending on a trade transaction type, it may be a price of an order, a deal or a position.
price_trigger	Stop limit order stop (activation) price (ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT).
price_sl	Stop Loss price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
price_tp	Take Profit price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
volume	Volume in lots. Depending on a trade transaction type, it may indicate the current volume of an order, a deal or a position.
position	The ticket of the position affected by the transaction.
position_by	The ticket of the opposite position. Used when closing a position by an opposite one, i.e. by a position of the same symbol that was opened in the opposite direction.

The essential parameter for received transaction analysis is its type specified in **type** field. For example, if a transaction is of [TRADE_TRANSACTION_REQUEST](#) type (a result of handling a trade request by the server has been received), the structure has only one field that is filled completely - **type**. Other fields are not analyzed. In this case, we may analyze two additional **request** and **result** parameters submitted to OnTradeTransaction() handler, as shown below.

Having data on a trading operation type, you can decide on the analysis of the current state of orders, positions and deals on a trading account. Remember that one trade request sent to the server from the terminal can generate several new transactions. The priority of their arrival at the terminal is not guaranteed.

MqlTradeTransaction structure is filled in different ways depending on a trade transaction type ([ENUM_TRADE_TRANSACTION_TYPE](#)):

TRADE_TRANSACTION_ORDER_* and TRADE_TRANSACTION_HISTORY_*

The following fields in MqlTradeTransaction structure are filled for trade transactions related to open orders handling (TRADE_TRANSACTION_ORDER_ADD, TRADE_TRANSACTION_ORDER_UPDATE and TRADE_TRANSACTION_ORDER_DELETE) and orders history (TRADE_TRANSACTION_HISTORY_ADD, TRADE_TRANSACTION_HISTORY_UPDATE, TRADE_TRANSACTION_HISTORY_DELETE):

- order - order ticket;
- symbol - order symbol name;
- type - trade transaction type;
- order_type - order type;
- orders_state - order current state;
- time_type - order expiration type;
- time_expiration - order expiration time (for orders having [ORDER_TIME_SPECIFIED](#) and [ORDER_TIME_SPECIFIED_DAY](#) expiration types);
- price - order price specified by a client;
- price_trigger - stop limit order stop price (only for [ORDER_TYPE_BUY_STOP_LIMIT](#) and [ORDER_TYPE_SELL_STOP_LIMIT](#));
- price_sl - Stop Loss order price (filled, if specified in the order);
- price_tp - Take Profit order price (filled, if specified in the order);
- volume - order current volume (unfilled). Initial order volume can be found in the orders history using [HistoryOrders*](#) function.
- position - the ticket of the position that was opened, modified or closed as a result of order execution. It is only filled for market orders, not filled for TRADE_TRANSACTION_ORDER_ADD.
- position_by - the ticket of the opposite position. It is only filled for the close by orders (to close a position by an opposite one).

TRADE_TRANSACTION_DEAL_*

The following fields in MqlTradeTransaction structure are filled for trade transactions related to deals handling (TRADE_TRANSACTION_DEAL_ADD, TRADE_TRANSACTION_DEAL_UPDATE and TRADE_TRANSACTION_DEAL_DELETE):

- deal - deal ticket;
- order - order ticket, based on which a deal has been performed;
- symbol - deal symbol name;
- type - trade transaction type;

- deal_type - deal type;
- price - deal price;
- price_sl - Stop Loss price (filled, if specified in the order, based on which a deal has been performed);
- price_tp - Take Profit price (filled, if specified in the order, based on which a deal has been performed);
- volume - deal volume in lots.
- position - the ticket of the position that was opened, modified or closed as a result of deal execution.
- position_by - the ticket of the opposite position. It is only filled for the out by deals (closing a position by an opposite one).

TRADE_TRANSACTION_POSITION

The following fields in MqlTradeTransaction structure are filled for trade transactions related to changing the positions not connected with deals execution (TRADE_TRANSACTION_POSITION):

- symbol - position symbol name;
- type - trade transaction type;
- deal_type - position type ([DEAL_TYPE_BUY](#) or [DEAL_TYPE_SELL](#));
- price - weighted average position open price;
- price_sl - Stop Loss price;
- price_tp - Take Profit price;
- volume - position volume in lots, if it has been changed.

Position change (adding, changing or closing), as a result of a deal execution, does not lead to the occurrence of TRADE_TRANSACTION_POSITION transaction.

TRADE_TRANSACTION_REQUEST

Only one field in MqlTradeTransaction structure is filled for trade transactions describing the fact that a trade request has been processed by a server and processing result has been received (TRADE_TRANSACTION_REQUEST):

- type - trade transaction type;

Only type field (trade transaction type) must be analyzed for such transactions. The second and third parameters of [OnTradeTransaction](#) function (request and result) must be analyzed for additional data.

Example:

```
input int MagicNumber=1234567;

//--- enable CTrade trading class and declare the variable of this class
#include <Trade\Trade.mqh>
CTrade trade;

//--- flags for installing and deleting the pending order
bool pending_done=false;
bool pending_deleted=false;

//--- pending order ticket will be stored here
ulong order_ticket;
```



```

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- set MagicNumber to mark all our orders
    trade.SetExpertMagicNumber(MagicNumber);
    //--- trade requests will be sent in asynchronous mode using OrderSendAsync() function
    trade.SetAsyncMode(true);
    //--- initialize the variable by zero
    order_ticket=0;
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
    //---installing a pending order
    if(!pending_done)
    {
        double ask=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
        double buy_stop_price=NormalizeDouble(ask+1000*_Point,(int)SymbolInfoInteger(_Symbol,SYMBOL_PRICE));
        bool res=trade.BuyStop(0.1,buy_stop_price,_Symbol);
        //--- if BuyStop() function performed successfully
        if(res)
        {
            pending_done=true;
            //--- get a result of the request sending from ctrade
            MqlTradeResult trade_result;
            trade.Result(trade_result);
            //--- get request_id for the sent request
            uint request_id=trade_result.request_id;
            Print("Request has been sent to set a pending order. Request_ID=",request_id);
            //--- storing the order ticket (will be zero if using the asynchronous mode)
            order_ticket=trade_result.order;
            //--- all is done, early exit from OnTick() handler
            return;
        }
    }
    //--- delete the pending order
    if(!pending_deleted)
    {
        //--- additional check
        if(pending_done && (order_ticket!=0))
        {
            //--- trying to delete the pending order
            bool res=trade.OrderDelete(order_ticket);
            Print("OrderDelete=",res);
        }
    }
}

```



```

    //--- when delete request is sent successfully
    if(res)
    {
        pending_deleted=true;
        //--- get the request execution result
        MqlTradeResult trade_result;
        trade.Result(trade_result);
        //--- take request ID from the result
        uint request_id=trade_result.request_id;
        //--- display in Journal
        Print("The request has been sent to delete a pending order #",order_ticket,
            ". Request_ID=",request_id,
            "\r\n");
        //--- fix the order ticket from the request result
        order_ticket=trade_result.order;
    }
}

//---
}

//+-----+
//| TradeTransaction function |
//+-----+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
{
    //--- get transaction type as enumeration value
    ENUM_TRADE_TRANSACTION_TYPE type=(ENUM_TRADE_TRANSACTION_TYPE)trans.type;
    //--- if the transaction is the request handling result, only its name is displayed
    if(type==TRADE_TRANSACTION_REQUEST)
    {
        Print(EnumToString(type));
        //--- display the handled request string name
        Print("-----RequestDescription\r\n",RequestDescription(request));
        //--- display request result description
        Print("-----ResultDescription\r\n",TradeResultDescription(result));
        //--- store the order ticket for its deletion at the next handling in OnTick()
        if(result.order!=0)
        {
            //--- delete this order by its ticket at the next OnTick() call
            order_ticket=result.order;
            Print(" Pending order ticket ",order_ticket,"\r\n");
        }
    }
    else // display the full description for transactions of another type
    //--- display description of the received transaction in the Journal
        Print("-----TransactionDescription\r\n",TransactionDescription(trans));
    //---

```



```

    }
//+-----+
//| Returns transaction textual description |
//+-----+
string TransactionDescription(const MqlTradeTransaction &trans)
{
//---
    string desc=EnumToString(trans.type)+"\r\n";
    desc+="Symbol: "+trans.symbol+"\r\n";
    desc+="Deal ticket: "+(string)trans.deal+"\r\n";
    desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
    desc+="Order ticket: "+(string)trans.order+"\r\n";
    desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
    desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
    desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
    desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
    desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
    desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
    desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
    desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
    desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
    desc+="Position: "+(string)trans.position+"\r\n";
    desc+="Position by: "+(string)trans.position_by+"\r\n";
//--- return the obtained string
    return desc;
}
//+-----+
//| Returns the trade request textual description |
//+-----+
string RequestDescription(const MqlTradeRequest &request)
{
//---
    string desc=EnumToString(request.action)+"\r\n";
    desc+="Symbol: "+request.symbol+"\r\n";
    desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
    desc+="Order ticket: "+(string)request.order+"\r\n";
    desc+="Order type: "+EnumToString(request.type)+"\r\n";
    desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
    desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
    desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
    desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
    desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
    desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
    desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
    desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
    desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
    desc+="Comment: "+request.comment+"\r\n";
//--- return the obtained string
    return desc;
}

```



```

    }
//+-----+
//| Returns the textual description of the request handling result |
//+-----+
string TradeResultDescription(const MqlTradeResult &result)
{
//---
    string desc="Retcode " + (string)result.retcode + "\r\n";
    desc+="Request ID: " + StringFormat("%d",result.request_id) + "\r\n";
    desc+="Order ticket: " + (string)result.order + "\r\n";
    desc+="Deal ticket: " + (string)result.deal + "\r\n";
    desc+="Volume: " + StringFormat("%G",result.volume) + "\r\n";
    desc+="Price: " + StringFormat("%G",result.price) + "\r\n";
    desc+="Ask: " + StringFormat("%G",result.ask) + "\r\n";
    desc+="Bid: " + StringFormat("%G",result.bid) + "\r\n";
    desc+="Comment: " + result.comment + "\r\n";
//--- return the obtained string
    return desc;
}

```

See also

[Trade Transaction Types](#), [OnTradeTransaction\(\)](#)

The Structure for Returning Current Prices (MqlTick)

This is a structure for storing the latest prices of the symbol. It is designed for fast retrieval of the most requested information about current prices.

```
struct MqlTick
{
    datetime    time;           // Time of the last prices update
    double      bid;           // Current Bid price
    double      ask;           // Current Ask price
    double      last;          // Price of the last deal (Last)
    ulong       volume;        // Volume for the current Last price
    long        time_msc;      // Time of a price last update in milliseconds
    uint        flags          // Tick flags
};
```

The variable of the MqlTick type allows obtaining values of Ask, Bid, Last and Volume within a single call of the [SymbolInfoTick\(\)](#) function.

The parameters of each tick are filled in regardless of whether there are changes compared to the previous tick. Thus, it is possible to find out a correct price for any moment in the past without the need to search for previous values at the tick history. For example, even if only a Bid price changes during a tick arrival, the structure still contains other parameters as well, including the previous Ask price, volume, etc.

You can analyze the tick flags to find out what data have been changed exactly:

- TICK_FLAG_BID - tick has changed a Bid price
- TICK_FLAG_ASK - a tick has changed an Ask price
- TICK_FLAG_LAST - a tick has changed the last deal price
- TICK_FLAG_VOLUME - a tick has changed a volume
- TICK_FLAG_BUY - a tick is a result of a buy deal
- TICK_FLAG_SELL - a tick is a result of a sell deal

Example:

```
void OnTick()
{
    MqlTick last_tick;
    //---
    if(SymbolInfoTick(Symbol(),last_tick))
    {
        Print(last_tick.time,": Bid = ",last_tick.bid,
            " Ask = ",last_tick.ask," Volume = ",last_tick.volume);
    }
    else Print("SymbolInfoTick() failed, error = ",GetLastError());
    //---
}
```

See also

[Structures and Classes](#), [CopyTicks\(\)](#), [SymbolInfoTick\(\)](#)

Codes of Errors and Warnings

This section contains the following descriptions:

- [Return codes of the trade server](#) - analyzing results of the [trade request](#) sent by function [OrderSend\(\)](#);
- [Compiler warnings](#) - codes of warning messages that appear at compilation (not errors);
- [Compilation errors](#) - codes of error messages at an unsuccessful attempt to compile;
- [Runtime errors](#) - error codes in the execution of mql5-programs, which can be obtained using the [GetLastError\(\)](#) function.

Return Codes of the Trade Server

All requests to execute trade operations are sent as a structure of a trade request [MqlTradeRequest](#) using function [OrderSend\(\)](#). The function execution result is placed to structure [MqlTradeResult](#), whose *retcode* field contains the trade server return code.

Code	Constant	Description
10004	TRADE_RETCODE_REQUOTE	Requote
10006	TRADE_RETCODE_REJECT	Request rejected
10007	TRADE_RETCODE_CANCEL	Request canceled by trader
10008	TRADE_RETCODE_PLACED	Order placed
10009	TRADE_RETCODE_DONE	Request completed
10010	TRADE_RETCODE_DONE_PARTIAL	Only part of the request was completed
10011	TRADE_RETCODE_ERROR	Request processing error
10012	TRADE_RETCODE_TIMEOUT	Request canceled by timeout
10013	TRADE_RETCODE_INVALID	Invalid request
10014	TRADE_RETCODE_INVALID_VOLUME	Invalid volume in the request
10015	TRADE_RETCODE_INVALID_PRICE	Invalid price in the request
10016	TRADE_RETCODE_INVALID_STOPS	Invalid stops in the request
10017	TRADE_RETCODE_TRADE_DISABLED	Trade is disabled
10018	TRADE_RETCODE_MARKET_CLOSED	Market is closed
10019	TRADE_RETCODE_NO_MONEY	There is not enough money to complete the request
10020	TRADE_RETCODE_PRICE_CHANGED	Prices changed
10021	TRADE_RETCODE_PRICE_OFF	There are no quotes to process the request
10022	TRADE_RETCODE_INVALID_EXPIRATION	Invalid order expiration date in the request
10023	TRADE_RETCODE_ORDER_CHANGED	Order state changed
10024	TRADE_RETCODE_TOO_MANY_REQUESTS	Too frequent requests

10025	TRADE_RETCODE_NO_CHANGES	No changes in request
10026	TRADE_RETCODE_SERVER_DISABLED_AT	Autotrading disabled by server
10027	TRADE_RETCODE_CLIENT_DISABLED_AT	Autotrading disabled by client terminal
10028	TRADE_RETCODE_LOCKED	Request locked for processing
10029	TRADE_RETCODE_FROZEN	Order or position frozen
10030	TRADE_RETCODE_INVALID_FILL	Invalid order filling type
10031	TRADE_RETCODE_CONNECTION	No connection with the trade server
10032	TRADE_RETCODE_ONLY_REAL	Operation is allowed only for live accounts
10033	TRADE_RETCODE_LIMIT_ORDERS	The number of pending orders has reached the limit
10034	TRADE_RETCODE_LIMIT_VOLUME	The volume of orders and positions for the symbol has reached the limit
10035	TRADE_RETCODE_INVALID_ORDER	Incorrect or prohibited order type
10036	TRADE_RETCODE_POSITION_CLOSED	Position with the specified POSITION_IDENTIFIER has already been closed
10038	TRADE_RETCODE_INVALID_CLOSE_VOLUME	A close volume exceeds the current position volume
10039	TRADE_RETCODE_CLOSE_ORDER_EXISTS	<p>A close order already exists for a specified position. This may happen when working in the hedging system:</p> <ul style="list-style-type: none"> • when attempting to close a position with an opposite one, while close orders for the position already exist • when attempting to fully or partially close a position if the total volume of the already present close orders and the newly placed one exceeds the current position volume
10040	TRADE_RETCODE_LIMIT_POSITIONS	The number of open positions simultaneously present on an

		<p>account can be limited by the server settings. After a limit is reached, the server returns the <code>TRADE_RETCODE_LIMIT_POSITIONS</code> error when attempting to place an order. The limitation operates differently depending on the position accounting type:</p> <ul style="list-style-type: none">• Netting — number of open positions is considered. When a limit is reached, the platform does not let placing new orders whose execution may increase the number of open positions. In fact, the platform allows placing orders only for the symbols that already have open positions. The current pending orders are not considered since their execution may lead to changes in the current positions but it cannot increase their number.• Hedging — pending orders are considered together with open positions, since a pending order activation always leads to opening a new position. When a limit is reached, the platform does not allow placing both new market orders for opening positions and pending orders.
--	--	--

Compiler Warnings

Compiler warnings are shown for informational purposes only and are not error messages.

Code	Description
21	Incomplete record of a date in the datetime string
22	Wrong number in the datetime string for the date. Requirements: Year $1970 \leq X \leq 3000$ Month $0 < X \leq 12$ Day $0 < X \leq 31/30/28 (29) \dots$
23	Wrong number of datetime string for time. Requirements: Hour $0 \leq X < 24$ Minute $0 \leq X < 60$
24	Invalid color in RGB format: one of RGB components is less than 0 or greater than 255
25	Unknown character of the escape sequences. Known: <code>\n \r \t \\ \" \' \X \x</code>
26	Too large volume of local variables (> 512Kb) of the function, reduce the number
29	Enumeration already defined (duplication) - members will be added to the first definition
30	Overriding macro
31	The variable is declared but is not used anywhere
32	Constructor must be of void type
33	Destructor must be of void type
34	Constant does not fit in the range of integers ($X > _UI64_MAX$ $X < _I64_MIN$) and will be converted to the double type
35	Too long HEX - more than 16 significant characters (senior nibbles are cut)
36	No nibbles in HEX string "0x"
37	No function - nothing to be performed
38	A non-initialized variable is used
41	Function has no body, and is not called

43	Possible loss of data at typecasting. Example: <code>int x = (double) z;</code>
44	Loss of accuracy (of data) when converting a constant. Example: <code>int x = M_PI</code>
45	Difference between the signs of operands in the operations of comparison. Example: <code>(char) c1 > (uchar) c2</code>
46	Problems with function importing - declaration of <code>#import</code> is required or import of functions is closed
47	Too large description - extra characters will not be included in the executable file
48	The number of indicator buffers declared is less than required
49	No color to plot a graphical series in the indicator
50	No graphical series to draw the indicator
51	'OnStart' handler function not found in the script
52	'OnStart' handler function is defined with wrong parameters
53	'OnStart' function can be defined only in a script
54	'OnInit' function is defined with wrong parameters
55	'OnInit' function is not used in scripts
56	'OnDeinit' function is defined with wrong parameters
57	'OnDeinit' function is not used in scripts
58	Two 'OnCalculate' functions are defined. OnCalculate () at one price array will be used
59	Overfilling detected when calculating a complex integer constant
60	Probably, the variable is not initialized .
61	This declaration makes it impossible to refer to the local variable declared on the specified line
62	This declaration makes it impossible to refer to the global variable declared on the specified line
63	Cannot be used for static allocated array

64	This variable declaration hides predefined variable
65	The value of the expression is always true/false
66	Using a variable or bool type expression in mathematical operations is unsafe
67	The result of applying the unary minus operator to an unsigned ulong type is undefined
68	The version specified in the #property version property is unacceptable for the Market section; the correct format of #property version id "XXX.YYY"
69	Empty controlled statement found
70	Invalid function return type or incorrect parameters during declaration of the event handler function
71	An implicit cast of structures to one type is required
72	This declaration makes direct access to the member of a class declared in the specified string impossible. Access will be possible only with the scope resolution operation ::
73	Binary constant is too big, high-order digits will be truncated
74	Parameter in the method of the inherited class has a different const modifier, the derived function has overloaded the parent function
75	Negative or too large shift value in shift bitwise operation , execution result is undefined
76	Function must return a value
77	void function returns a value
78	Not all control paths return a value
79	Expressions are not allowed on a global scope
80	Check operator precedence for possible error; use parentheses to clarify precedence
81	Two OnCalculate() are defined. OHLC version will be used
82	Struct has no members, size assigned to 1 byte
83	Return value of the function should be checked

84	Resource indicator is compiled for debugging. That slows down the performance. Please recompile the indicator to increase performance
85	Too great character code in the string, must be in the range 0 to 65535
86	Unrecognized character in the string
87	No indicator window property (setting the display in the main window or a subwindow) is defined. Property #property indicator_chart_window is applied

Compilation Errors

MetaEditor 5 shows error messages about the program errors detected by the built-in compiler during compilation. The list of these errors is given below in table. To compile a source code into an executable one, press **F7**. Programs that contain errors cannot be compiled until the errors identified by the compiler are eliminated.

Code	Description
100	File reading error
101	Error of opening an *. EX5 for writing
103	Not enough free memory to complete compilation
104	Empty syntactic unit unrecognized by compiler
105	Incorrect file name in #include
106	Error accessing a file in #include (probably the file does not exist)
108	Inappropriate name for #define
109	Unknown command of preprocessor (valid #include, #define, #property, #import)
110	Symbol unknown to compiler
111	Function not implemented (description is present, but no body)
112	Double quote (") omitted
113	Opening angle bracket (<) or double quote (") omitted
114	Single quote (') omitted
115	Closing angle bracket ">" omitted
116	Type not specified in declaration
117	No return operator or return is found not in all branches of the implementation
118	Opening bracket of call parameters was expected
119	Error writing EX5
120	Invalid access to an array
121	The function is not of void type and the return operator must return a value
122	Incorrect declaration of the destructor

123	Colon ":" is missing
124	Variable is already declared
125	Variable with such identifier already declared
126	Variable name is too long (> 250 characters)
127	Structure with such identifier already defined
128	Structure is not defined
129	Structure member with the same name already defined
130	No such structure member
131	Breached pairing of brackets
132	Opening parenthesis "(" expected
133	Unbalanced braces (no "}")
134	Difficult to compile (too much branching, internal stack levels are overfilled)
135	Error of file opening for reading
136	Not enough memory to download the source file into memory
137	Variable is expected
138	Reference cannot be initialized
140	Assignment expected (appears at declaration)
141	Opening brace "{" expected
142	Parameter can be a dynamic array only
143	Use of "void" type is unacceptable
144	No pair for ")" or "]", i.e. "(or" [" is absent
145	No pair for "(or" [", i.e. ") "or"] " is absent
146	Incorrect array size
147	Too many parameters (> 64)
149	This token is not expected here
150	Invalid use of operation (invalid operands)
151	Expression of void type not allowed
152	Operator is expected
153	Misuse of break
154	Semicolon ";" expected

155	Comma "," expected
156	Must be a class type, not struct
157	Expression is expected
158	"non HEX character" found in HEX or too long number (number of digits> 511)
159	String-constant has more than 65534 characters
160	Function definition is unacceptable here
161	Unexpected end of program
162	Forward declaration is prohibited for structures
163	Function with this name is already defined and has another return type
164	Function with this name is already defined and has a different set of parameters
165	Function with this name is already defined and implemented
166	Function overload for this call was not found
167	Function with a return value of void type cannot return a value
168	Function is not defined
170	Value is expected
171	In <i>case</i> expression only integer constants are valid
172	The value of <i>case</i> in this <i>switch</i> is already used
173	Integer is expected
174	In <i>#import</i> expression file name is expected
175	Expressions are not allowed on global level
176	Omitted parenthesis ")" before ";"
177	To the left of equality sign a variable is expected
178	The result of expression is not used
179	Declaring of variables is not allowed in <i>case</i>
180	Implicit conversion from a string to a number
181	Implicit conversion of a number to a string

182	Ambiguous call of an overloaded function (several overloads fit)
183	Illegal <i>else</i> without proper <i>if</i>
184	Invalid <i>case</i> or <i>default</i> without a <i>switch</i>
185	Inappropriate use of ellipsis
186	The initializing sequence has more elements than the initialized variable
187	A constant for <i>case</i> expected
188	A constant expression required
189	A constant variable cannot be changed
190	Closing bracket or a comma is expected (declaring array member)
191	Enumerator identifier already defined
192	Enumeration cannot have access modifiers (const, extern, static)
193	Enumeration member already declared with a different value
194	There is a variable defined with the same name
195	There is a structure defined with the same name
196	Name of enumeration member expected
197	Integer expression expected
198	Division by zero in constant expression
199	Wrong number of parameters in the function
200	Parameter by reference must be a variable
201	Variable of the same type to pass by reference expected
202	A constant variable cannot be passed by a non-constant reference
203	Requires a positive integer constant
204	Failed to access protected class member
205	Import already defined in another way
208	Executable file not created
209	'OnCalculate' entry point not found for the indicator

210	The continue operation can be used only inside a loop
211	Error accessing private (closed) class member
213	Method of structure or class is not declared
214	Error accessing private (closed) class method
216	Copying of structures with objects is not allowed
218	Index out of array range
219	Array initialization in structure or class declaration not allowed
220	Class constructor cannot have parameters
221	Class destructor can not have parameters
222	Class method or structure with the same name and parameters have already been declared
223	Operand expected
224	Class method or structure with the same name exists, but with different parameters (declaration!=implementation)
225	Imported function is not described
226	ZeroMemory() is not allowed for objects with protected members or inheritance
227	Ambiguous call of the overloaded function (exact match of parameters for several overloads)
228	Variable name expected
229	A reference cannot be declared in this place
230	Already used as the enumeration name
232	Class or structure expected
235	Cannot call 'delete' operator to delete the array
236	Operator ' while' expected
237	Operator 'delete' must have a pointer
238	There is 'default' for this 'switch' already
239	Syntax error
240	Escape-sequence can occur only in strings (starts with '\')

241	Array required - square bracket '[' does not apply to an array, or non arrays are passed as array parameters
242	Can not be initialized through the initialization sequence
243	Import is not defined
244	Optimizer error on the syntactic tree
245	Declared too many structures (try to simplify the program)
246	Conversion of the parameter is not allowed
247	Incorrect use of the 'delete' operator
248	It's not allowed to declare a pointer to a reference
249	It's not allowed to declare a reference to a reference
250	It's not allowed to declare a pointer to a pointer
251	Structure declaration in the list of parameter is not allowed
252	Invalid operation of typecasting
253	A pointer can be declared only for a class or structure
256	Undeclared identifier
257	Executable code optimizer error
258	Executable code generation error
260	Invalid expression for the 'switch' operator
261	Pool of string constants overfilled, simplify program
262	Cannot convert to enumeration
263	Do not use 'virtual' for data (members of a class or structure)
264	Cannot call protected method of class
265	Overridden virtual functions return a different type
266	Class cannot be inherited from a structure
267	Structure cannot be inherited from a class

268	Constructor cannot be virtual (<i>virtual</i> specifier is not allowed)
269	Method of structure cannot be virtual
270	Function must have a body
271	Overloading of system functions (terminal functions) is prohibited
272	<i>Const</i> specifier is invalid for functions that are not members of a class or structure
274	Not allowed to change class members in constant method
276	Inappropriate initialization sequence
277	Missed default value for the parameter (specific declaration of default parameters)
278	Overriding the default parameter (different values in declaration and implementation)
279	Not allowed to call non-constant method for a constant object
280	An object is necessary for accessing members (a dot for a non class/structure is specified)
281	The name of an already declared structure cannot be used in declaration
284	Unauthorized conversion (at closed inheritance)
285	Structures and arrays cannot be used as input variables
286	<i>Const</i> specifier is not valid for constructor/destructor
287	Incorrect string expression for a datetime
288	Unknown property (#property)
289	Incorrect value of a property
290	Invalid index for a property in #property
291	Call parameter omitted - <func (x,)>
293	Object must be passed by reference
294	Array must be passed by reference
295	Function was declared as exportable
296	Function was not declared as exportable
297	It is prohibited to export imported function

298	Imported function cannot have this parameter (prohibited to pass a pointer, class or structure containing a dynamic array, pointer, class, etc.)
299	Must be a class
300	#import was not closed
302	Type mismatch
303	Extern variable is already initialized
304	No exported function or entry point found
305	Explicit constructor call is not allowed
306	Method was declared as constant
307	Method was not declared as constant
308	Incorrect size of the resource file
309	Incorrect resource name
310	Resource file opening error
311	Resource file reading error
312	Unknown resource type
313	Incorrect path to the resource file
314	The specified resource name is already used
315	Argument expected for the function-like macro
316	Unexpected symbol in macro definition
317	Error in formal parameters of the macro
318	Invalid number of parameters for a macro
319	Too many parameters for a macro
320	Too complex, simplify the macro
321	Parameter for EnumToString() can be only an enumeration
322	The resource name is too long
323	Unsupported image format (only BMP with 24 or 32 bit color depth is supported)
324	An array cannot be declared in operator
325	The function can be declared only in the global scope
326	The declaration is not allowed for the current scope

327	Initialization of static variables with the values of local variables is not allowed
328	Illegal declaration of an array of objects that do not have a default constructor
329	Initialization list allowed only for constructors
330	No function definition after initialization list
331	Initialization list is empty
332	Array initialization in a constructor is not allowed
333	Initializing members of a parent class in the initialization list is not allowed
334	Expression of the integer type expected
335	Memory required for the array exceeds the maximum value
336	Memory required for the structure exceeds the maximum value
337	Memory required for the variables declared on the global level exceeds the maximum value
338	Memory required for local variables exceeds the maximum value
339	Constructor not defined
340	Invalid name of the icon file
341	Could not open the icon file at the specified path
342	The icon file is incorrect and is not of the ICO format
343	Reinitialization of a member in a class/structure constructor using the initialization list
344	Initialization of static members in the constructor initialization list is not allowed
345	Initialization of a non-static member of a class/structure on a global level is not allowed
346	The name of the class/structure method matches the name of an earlier declared member
347	The name of the class/structure member matches the name of an earlier declared

	method
348	Virtual function cannot be declared as static
349	The const modifier is not allowed for static functions
350	Constructor or destructor cannot be static
351	Non-static member/method of a class or a structure cannot be accessed from a static function
352	An overload operation (+,-,[],++,-- etc.) is expected after the operator keyword
353	Not all operations can be overloaded in MQL5
354	Definition does not match declaration
355	An invalid number of parameters is specified for the operator
356	Event handling function not found
357	Method cannot be exported
358	A pointer to the constant object cannot be normalized by a non-constant object
359	Class templates are not supported yet
360	Function template overload is not supported yet
361	Function template cannot be applied
362	Ambiguous parameter in function template (several parameter types can be applied)
363	Unable to determine the parameter type, by which the function template argument should be normalized
364	Incorrect number of parameters in the function template
365	Function template cannot be virtual
366	Function templates cannot be exported
367	Function templates cannot be imported
368	Structures containing the objects are not allowed
369	String arrays and structures containing the objects are not allowed

370	A static class/structure member must be explicitly initialized
371	Compiler limitation: the string cannot contain more than 65 535 characters
372	Inconsistent #ifdef/#endif
373	Object of class cannot be returned, copy constructor not found
374	Non-static members and methods cannot be used
375	OnTesterInit() impossible to use without OnTesterDeinit()
376	Redefinition of formal parameter '%s'
377	Macro __FUNCSIG__ and __FUNCTION__ cannot appear outside of a function body
378	Invalid returned type. For example, this error will be produced for functions imported from DLL that return structure or pointer.
379	Template usage error
380	Not used
381	Illegal syntax when declaring pure virtual function, only "=NULL" or "=0" are allowed
382	Only virtual functions can be declared with the pure-specifier ("=NULL" or "=0")
383	Abstract class cannot be instantiated
384	A pointer to a user-defined type should be applied as a target type for dynamic casting using the dynamic_cast operator
385	"Pointer to function" type is expected
386	Pointers to methods are not supported
387	Error - cannot define the type of a pointer to function
388	Type cast is not available due to private inheritance
389	A variable with const modifier should be initialized during declaration
393	Only methods with public access can be declared in an interface

394	Invalid nesting of an interface inside of another interface
395	An interface can only be derived from another interface
396	An interface is expected
397	Interfaces only support public inheritance
398	An interface cannot contain members
399	Interface objects cannot be created directly, only use inheritance

Runtime Errors

[GetLastError\(\)](#) is the function that returns the last error code that is stored in the predefined variable [_LastError](#). This value can be reset to zero by the [ResetLastError\(\)](#) function.

Constant	Code	Description
ERR_SUCCESS	0	The operation completed successfully
ERR_INTERNAL_ERROR	4001	Unexpected internal error
ERR_WRONG_INTERNAL_PARAMETER	4002	Wrong parameter in the inner call of the client terminal function
ERR_INVALID_PARAMETER	4003	Wrong parameter when calling the system function
ERR_NOT_ENOUGH_MEMORY	4004	Not enough memory to perform the system function
ERR_STRUCT_WITHOBJECTS_ORCLASS	4005	The structure contains objects of strings and/or dynamic arrays and/or structure of such objects and/or classes
ERR_INVALID_ARRAY	4006	Array of a wrong type, wrong size, or a damaged object of a dynamic array
ERR_ARRAY_RESIZE_ERROR	4007	Not enough memory for the relocation of an array, or an attempt to change the size of a static array
ERR_STRING_RESIZE_ERROR	4008	Not enough memory for the relocation of string
ERR_NOTINITIALIZED_STRING	4009	Not initialized string
ERR_INVALID_DATETIME	4010	Invalid date and/or time
ERR_ARRAY_BAD_SIZE	4011	Requested array size exceeds 2 GB
ERR_INVALID_POINTER	4012	Wrong pointer
ERR_INVALID_POINTER_TYPE	4013	Wrong type of pointer
ERR_FUNCTION_NOT_ALLOWED	4014	Function is not allowed for call
ERR_RESOURCE_NAME_DUPLICATED	4015	The names of the dynamic and the static resource match

ERR_RESOURCE_NOT_FOUND	4016	Resource with this name has not been found in EX5
ERR_RESOURCE_UNSUPPORTED_TYPE	4017	Unsupported resource type or its size exceeds 16 Mb
ERR_RESOURCE_NAME_IS_TOO_LONG	4018	The resource name exceeds 63 characters
Charts		
ERR_CHART_WRONG_ID	4101	Wrong chart ID
ERR_CHART_NO_REPLY	4102	Chart does not respond
ERR_CHART_NOT_FOUND	4103	Chart not found
ERR_CHART_NO_EXPERT	4104	No Expert Advisor in the chart that could handle the event
ERR_CHART_CANNOT_OPEN	4105	Chart opening error
ERR_CHART_CANNOT_CHANGE	4106	Failed to change chart symbol and period
ERR_CHART_WRONG_PARAMETER	4107	Error value of the parameter for the function of working with charts
ERR_CHART_CANNOT_CREATE_TIMER	4108	Failed to create timer
ERR_CHART_WRONG_PROPERTY	4109	Wrong chart property ID
ERR_CHART_SCREENSHOT_FAILED	4110	Error creating screenshots
ERR_CHART_NAVIGATE_FAILED	4111	Error navigating through chart
ERR_CHART_TEMPLATE_FAILED	4112	Error applying template
ERR_CHART_WINDOW_NOT_FOUND	4113	Subwindow containing the indicator was not found
ERR_CHART_INDICATOR_CANNOT_ADD	4114	Error adding an indicator to chart
ERR_CHART_INDICATOR_CANNOT_DEL	4115	Error deleting an indicator from the chart
ERR_CHART_INDICATOR_NOT_FOUND	4116	Indicator not found on the specified chart
Graphical Objects		

ERR_OBJECT_ERROR	4201	Error working with a graphical object
ERR_OBJECT_NOT_FOUND	4202	Graphical object was not found
ERR_OBJECT_WRONG_PROPERTY	4203	Wrong ID of a graphical object property
ERR_OBJECT_GETDATE_FAILED	4204	Unable to get date corresponding to the value
ERR_OBJECT_GETVALUE_FAILED	4205	Unable to get value corresponding to the date
MarketInfo		
ERR_MARKET_UNKNOWN_SYMBOL	4301	Unknown symbol
ERR_MARKET_NOT_SELECTED	4302	Symbol is not selected in MarketWatch
ERR_MARKET_WRONG_PROPERTY	4303	Wrong identifier of a symbol property
ERR_MARKET_LASTTIME_UNKNOWN	4304	Time of the last tick is not known (no ticks)
ERR_MARKET_SELECT_ERROR	4305	Error adding or deleting a symbol in MarketWatch
History Access		
ERR_HISTORY_NOT_FOUND	4401	Requested history not found
ERR_HISTORY_WRONG_PROPERTY	4402	Wrong ID of the history property
ERR_HISTORY_TIMEOUT	4403	Exceeded history request timeout
ERR_HISTORY_BARS_LIMIT	4404	Number of requested bars limited by terminal settings
ERR_HISTORY_LOAD_ERRORS	4405	Multiple errors when loading history
ERR_HISTORY_SMALL_BUFFER	4407	Receiving array is too small to store all requested data
Global Variables		
ERR_GLOBALVARIABLE_NOT_FOUND	4501	Global variable of the client terminal is not found
ERR_GLOBALVARIABLE_EXISTS	4502	Global variable of the client terminal with the same name already exists

ERR_MAIL_SEND_FAILED	4510	Email sending failed
ERR_PLAY_SOUND_FAILED	4511	Sound playing failed
ERR_MQL5_WRONG_PROPERTY	4512	Wrong identifier of the program property
ERR_TERMINAL_WRONG_PROPERTY	4513	Wrong identifier of the terminal property
ERR_FTP_SEND_FAILED	4514	File sending via ftp failed
ERR_NOTIFICATION_SEND_FAILED	4515	Failed to send a notification
ERR_NOTIFICATION_WRONG_PARAMETER	4516	Invalid parameter for sending a notification - an empty string or NULL has been passed to the SendNotification() function
ERR_NOTIFICATION_WRONG_SETTINGS	4517	Wrong settings of notifications in the terminal (ID is not specified or permission is not set)
ERR_NOTIFICATION_TOO_FREQUENT	4518	Too frequent sending of notifications
ERR_FTP_NO_SERVER	4519	FTP server is not specified
ERR_FTP_NO_LOGIN	4520	FTP login is not specified
ERR_FTP_FILE_ERROR	4521	File not found in the MQL5\Files directory to send on FTP server
ERR_FTP_CONNECT_FAILED	4522	FTP connection failed
ERR_FTP_CHANGEDIR	4523	FTP path not found on server
ERR_FTP_CLOSED	4524	FTP connection closed
Custom Indicator Buffers		
ERR_BUFFERS_NO_MEMORY	4601	Not enough memory for the distribution of indicator buffers
ERR_BUFFERS_WRONG_INDEX	4602	Wrong indicator buffer index
Custom Indicator Properties		
ERR_CUSTOM_WRONG_PROPERTY	4603	Wrong ID of the custom indicator property
Account		

ERR_ACCOUNT_WRONG_PROPERTY	4701	Wrong account property ID
ERR_TRADE_WRONG_PROPERTY	4751	Wrong trade property ID
ERR_TRADE_DISABLED	4752	Trading by Expert Advisors prohibited
ERR_TRADE_POSITION_NOT_FOUND	4753	Position not found
ERR_TRADE_ORDER_NOT_FOUND	4754	Order not found
ERR_TRADE_DEAL_NOT_FOUND	4755	Deal not found
ERR_TRADE_SEND_FAILED	4756	Trade request sending failed
Indicators		
ERR_INDICATOR_UNKNOWN_SYMBOL	4801	Unknown symbol
ERR_INDICATOR_CANNOT_CREATE	4802	Indicator cannot be created
ERR_INDICATOR_NO_MEMORY	4803	Not enough memory to add the indicator
ERR_INDICATOR_CANNOT_APPLY	4804	The indicator cannot be applied to another indicator
ERR_INDICATOR_CANNOT_ADD	4805	Error applying an indicator to chart
ERR_INDICATOR_DATA_NOT_FOUND	4806	Requested data not found
ERR_INDICATOR_WRONG_HANDLE	4807	Wrong indicator handle
ERR_INDICATOR_WRONG_PARAMETERS	4808	Wrong number of parameters when creating an indicator
ERR_INDICATOR_PARAMETERS_MISSING	4809	No parameters when creating an indicator
ERR_INDICATOR_CUSTOM_NAME	4810	The first parameter in the array must be the name of the custom indicator
ERR_INDICATOR_PARAMETER_TYPE	4811	Invalid parameter type in the array when creating an indicator

ERR_INDICATOR_WRONG_INDEX	4812	Wrong index of the requested indicator buffer
Depth of Market		
ERR_BOOKS_CANNOT_ADD	4901	Depth Of Market can not be added
ERR_BOOKS_CANNOT_DELETE	4902	Depth Of Market can not be removed
ERR_BOOKS_CANNOT_GET	4903	The data from Depth Of Market can not be obtained
ERR_BOOKS_CANNOT_SUBSCRIBE	4904	Error in subscribing to receive new data from Depth Of Market
File Operations		
ERR_TOO_MANY_FILES	5001	More than 64 files cannot be opened at the same time
ERR_WRONG_FILENAME	5002	Invalid file name
ERR_TOO_LONG_FILENAME	5003	Too long file name
ERR_CANNOT_OPEN_FILE	5004	File opening error
ERR_FILE_CACHEBUFFER_ERROR	5005	Not enough memory for cache to read
ERR_CANNOT_DELETE_FILE	5006	File deleting error
ERR_INVALID_FILEHANDLE	5007	A file with this handle was closed, or was not opening at all
ERR_WRONG_FILEHANDLE	5008	Wrong file handle
ERR_FILE_NOTTOWRITE	5009	The file must be opened for writing
ERR_FILE_NOTTOREAD	5010	The file must be opened for reading
ERR_FILE_NOTBIN	5011	The file must be opened as a binary one
ERR_FILE_NOTTXT	5012	The file must be opened as a text
ERR_FILE_NOTTXTORCSV	5013	The file must be opened as a text or CSV
ERR_FILE_NOTCSV	5014	The file must be opened as CSV
ERR_FILE_READERROR	5015	File reading error

ERR_FILE_BINSTRINGSIZE	5016	String size must be specified, because the file is opened as binary
ERR_INCOMPATIBLE_FILE	5017	A text file must be for string arrays, for other arrays - binary
ERR_FILE_IS_DIRECTORY	5018	This is not a file, this is a directory
ERR_FILE_NOT_EXIST	5019	File does not exist
ERR_FILE_CANNOT_REWRITE	5020	File can not be rewritten
ERR_WRONG_DIRECTORYNAME	5021	Wrong directory name
ERR_DIRECTORY_NOT_EXIST	5022	Directory does not exist
ERR_FILE_ISNOT_DIRECTORY	5023	This is a file, not a directory
ERR_CANNOT_DELETE_DIRECTORY	5024	The directory cannot be removed
ERR_CANNOT_CLEAN_DIRECTORY	5025	Failed to clear the directory (probably one or more files are blocked and removal operation failed)
ERR_FILE_WRITEERROR	5026	Failed to write a resource to a file
ERR_FILE_ENDOFFILE	5027	Unable to read the next piece of data from a CSV file (FileReadString, FileReadNumber, FileReadDatetime, FileReadBool), since the end of file is reached
String Casting		
ERR_NO_STRING_DATE	5030	No date in the string
ERR_WRONG_STRING_DATE	5031	Wrong date in the string
ERR_WRONG_STRING_TIME	5032	Wrong time in the string
ERR_STRING_TIME_ERROR	5033	Error converting string to date
ERR_STRING_OUT_OF_MEMORY	5034	Not enough memory for the string
ERR_STRING_SMALL_LEN	5035	The string length is less than expected

ERR_STRING_TOO_BIGNUMBER	5036	Too large number, more than ULONG_MAX
ERR_WRONG_FORMATSTRING	5037	Invalid format string
ERR_TOO_MANY_FORMATTERS	5038	Amount of format specifiers more than the parameters
ERR_TOO_MANY_PARAMETERS	5039	Amount of parameters more than the format specifiers
ERR_WRONG_STRING_PARAMETER	5040	Damaged parameter of string type
ERR_STRINGPOS_OUTOFRANGE	5041	Position outside the string
ERR_STRING_ZEROADDED	5042	0 added to the string end, a useless operation
ERR_STRING_UNKNOWNTYPE	5043	Unknown data type when converting to a string
ERR_WRONG_STRING_OBJECT	5044	Damaged string object
Operations with Arrays		
ERR_INCOMPATIBLE_ARRAYS	5050	Copying incompatible arrays. String array can be copied only to a string array, and a numeric array - in numeric array only
ERR_SMALL_ASERIES_ARRAY	5051	The receiving array is declared as AS_SERIES, and it is of insufficient size
ERR_SMALL_ARRAY	5052	Too small array, the starting position is outside the array
ERR_ZEROSIZE_ARRAY	5053	An array of zero length
ERR_NUMBER_ARRAYS_ONLY	5054	Must be a numeric array
ERR_ONEDIM_ARRAYS_ONLY	5055	Must be a one-dimensional array
ERR_SERIES_ARRAY	5056	Timeseries cannot be used
ERR_DOUBLE_ARRAY_ONLY	5057	Must be an array of type double
ERR_FLOAT_ARRAY_ONLY	5058	Must be an array of type float
ERR_LONG_ARRAY_ONLY	5059	Must be an array of type long
ERR_INT_ARRAY_ONLY	5060	Must be an array of type int
ERR_SHORT_ARRAY_ONLY	5061	Must be an array of type short

ERR_CHAR_ARRAY_ONLY	5062	Must be an array of type char
Operations with OpenCL		
ERR_OPENCL_NOT_SUPPORTED	5100	OpenCL functions are not supported on this computer
ERR_OPENCL_INTERNAL	5101	Internal error occurred when running OpenCL
ERR_OPENCL_INVALID_HANDLE	5102	Invalid OpenCL handle
ERR_OPENCL_CONTEXT_CREATE	5103	Error creating the OpenCL context
ERR_OPENCL_QUEUE_CREATE	5104	Failed to create a run queue in OpenCL
ERR_OPENCL_PROGRAM_CREATE	5105	Error occurred when compiling an OpenCL program
ERR_OPENCL_TOO_LONG_KERNEL_NAME	5106	Too long kernel name (OpenCL kernel)
ERR_OPENCL_KERNEL_CREATE	5107	Error creating an OpenCL kernel
ERR_OPENCL_SET_KERNEL_PARAMETER	5108	Error occurred when setting parameters for the OpenCL kernel
ERR_OPENCL_EXECUTE	5109	OpenCL program runtime error
ERR_OPENCL_WRONG_BUFFER_SIZE	5110	Invalid size of the OpenCL buffer
ERR_OPENCL_WRONG_BUFFER_OFFSET	5111	Invalid offset in the OpenCL buffer
ERR_OPENCL_BUFFER_CREATE	5112	Failed to create an OpenCL buffer
Operations with WebRequest		
ERR_WEBREQUEST_INVALID_ADDRESS	5200	Invalid URL
ERR_WEBREQUEST_CONNECT_FAILED	5201	Failed to connect to specified URL
ERR_WEBREQUEST_TIMEOUT	5202	Timeout exceeded
ERR_WEBREQUEST_REQUEST_FAILED	5203	HTTP request failed
User-Defined Errors		

ERR_USER_ERROR_FIRST	65536	User defined errors start with this code
----------------------	-------	--

See also

[Trade Server Return Codes](#)

Input and Output Constants

Constants:

- [File opening flags](#)
- [File properties](#)
- [Positioning inside a file](#)
- [Code page usage](#)
- [MessageBox](#)

File Opening Flags

File opening flag values specify the file access mode. Flags are defined as follows:

Identifier	Value	Description
FILE_READ	1	File is opened for reading. Flag is used in FileOpen() . When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_WRITE	2	File is opened for writing. Flag is used in FileOpen() . When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_BIN	4	Binary read/write mode (without string to string conversion). Flag is used in FileOpen() .
FILE_CSV	8	CSV file (all its elements are converted to strings of the appropriate type, Unicode or ANSI, and separated by separator). Flag is used in FileOpen() .
FILE_TXT	16	Simple text file (the same as csv file, but without taking into account the separators). Flag is used in FileOpen() .
FILE_ANSI	32	Strings of ANSI type (one byte symbols). Flag is used in FileOpen() .
FILE_UNICODE	64	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen() .
FILE_SHARE_READ	128	Shared access for reading from several programs. Flag is used in FileOpen() , but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_SHARE_WRITE	256	Shared access for writing from several programs. Flag is used in FileOpen() , but it does not

		replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_REWRITE	512	Possibility for the file rewrite using functions FileCopy() and FileMove() . The file should exist or should be opened for writing, otherwise the file will not be opened.
FILE_COMMON	4096	The file path in the common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen() , FileCopy() , FileMove() and in FileExists() functions.

One or several flags can be specified when opening a file. This is a combination of flags. The combination of flags is written using the sign of logical OR (|), which is positioned between enumerated flags. For example, to open a file in CSV format for reading and writing at the same time, specify the combination FILE_READ|FILE_WRITE|FILE_CSV.

Example:

```
int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

There are some specific features of work when you specify read and write flags:

- If FILE_READ is specified, an attempt is made to open an existing file. If a file does not exist, file opening fails, a new file is not created.
- FILE_READ|FILE_WRITE - a new file is created if the file with the specified name does not exist.
- FILE_WRITE - the file is created again with a zero size.

When opening a file, specification of FILE_WRITE and/or FILE_READ is required.

Flags that define the type of reading of an open file possess priority. The highest flag is FILE_CSV, then goes FILE_BIN, and FILE_TXT is of lowest priority. Thus, if several flags are specified at the same time, (FILE_TXT|FILE_CSV or FILE_TXT|FILE_BIN or FILE_BIN|FILE_CSV), the flag with the highest priority will be used.

Flags that define the type of encoding also have priority. FILE_UNICODE is of a higher priority than FILE_ANSI. So if you specify combination FILE_UNICODE|FILE_ANSI, flag FILE_UNICODE will be used.

If neither FILE_UNICODE nor FILE_ANSI is indicated, FILE_UNICODE is implied. If neither FILE_CSV, nor FILE_BIN, nor FILE_TXT is specified, FILE_CSV is implied.

If a file is opened for reading as a text file (FILE_TXT or FILE_CSV), and at the file beginning a special two-byte indication `0xff,0xfe` is found, the encoding flag will be FILE_UNICODE, even if FILE_ANSI is specified.

See also

[File Functions](#)

File Properties

The [FileGetInteger\(\)](#) function is used for obtaining file properties. The identifier of the required property from the ENUM_FILE_PROPERTY_INTEGER enumeration is passed to it during call.

ENUM_FILE_PROPERTY_INTEGER

ID	ID description
FILE_EXISTS	Check the existence
FILE_CREATE_DATE	Date of creation
FILE_MODIFY_DATE	Date of the last modification
FILE_ACCESS_DATE	Date of the last access to the file
FILE_SIZE	File size in bytes
FILE_POSITION	Position of a pointer in the file
FILE_END	Get the end of file sign
FILE_LINE_END	Get the end of line sign
FILE_IS_COMMON	The file is opened in a shared folder of all terminals (see FILE_COMMON)
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)
FILE_IS_READABLE	The opened file is readable (see FILE_READ)
FILE_IS_WRITABLE	The opened file is writable (see FILE_WRITE)

The [FileGetInteger\(\)](#) function has two different options of call. In the first option, for getting properties of a file, its handle is specified, which is obtained while opening the file using the [FileOpen\(\)](#) function. This option allows getting all properties of a file.

The second option of the [FileGetInteger\(\)](#) function returns values of file properties by the file name. Using this option, only the following general properties can be obtained:

- FILE_EXISTS - existence of a file with a specified name
- FILE_CREATE_DATE - date of creation of the file with the specified name
- FILE_MODIFY_DATE - date of modification of the file with the specified name
- FILE_ACCESS_DATE - date of the last access to the file with the specified name
- FILE_SIZE - size of the file with the specified name

When trying to get properties other than specified above, the second option of FileGetInteger() call will return an error.

Positioning Inside a File

Most of [file functions](#) are associated with data read/write operations. At the same time, using the [FileSeek\(\)](#) you can specify the position of a file pointer to a position inside the file, from which the next read or write operation will be performed. The ENUM_FILE_POSITION enumeration contains valid pointer positions, relative to which you can specify the shift in bytes for the next operation.

ENUM_FILE_POSITION

Identifier	Description
SEEK_SET	File beginning
SEEK_CUR	Current position of a file pointer
SEEK_END	File end

See also

[FileIsEnding](#), [FileIsLineEnding](#)

Using a Codepage in String Conversion Operations

When converting [string](#) variables into arrays of [char type](#) and back, the encoding that by default corresponds to the current ANSI of Windows operating system (CP_ACP) is used in MQL5. If you want to specify a different type of encoding, it can be set as additional parameter for the [CharArrayToString\(\)](#), [StringToCharArray\(\)](#) and [FileOpen\(\)](#) functions.

The table lists the built-in constants for some of the most popular code pages. Not mentioned code pages can be specified by a code corresponding to the page.

Built-in Constants of Codepages

Constant	Value	Description
CP_ACP	0	The current Windows ANSI code page.
CP_OEMCP	1	The current system OEM code page.
CP_MACCP	2	The current system Macintosh code page. Note: This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unicode for encoding.
CP_THREAD_ACP	3	The Windows ANSI code page for the current thread.
CP_SYMBOL	42	Symbol code page
CP_UTF7	65000	UTF-7 code page.
CP_UTF8	65001	UTF-8 code page.

See also

[Client Terminal Properties](#)

Constants of the MessageBox Dialog Window

This section contains return codes of the [MessageBox\(\)](#) function. If a message window has a Cancel button, the function returns IDCANCEL, in case if the ESC key or the Cancel button is pressed. If there is no Cancel button in the message window, the pressing of ESC does not give any effect.

Constant	Value	Description
IDOK	1	"OK" button has been pressed
IDCANCEL	2	"Cancel" button has been pressed
IDABORT	3	"Abort" button has been pressed
IDRETRY	4	"Retry" button has been pressed
IDIGNORE	5	"Ignore" button has been pressed
IDYES	6	"Yes" button has been pressed
IDNO	7	"No" button has been pressed
IDTRYAGAIN	10	"Try Again" button has been pressed
IDCONTINUE	11	"Continue" button has been pressed

The main flags of the [MessageBox\(\)](#) function define contents and behavior of the dialog window. This value can be a combination of the following flag groups:

Constant	Value	Description
MB_OK	0x00000000	Message window contains only one button: OK. Default
MB_OKCANCEL	0x00000001	Message window contains two buttons: OK and Cancel
MB_ABORTRETRYIGNORE	0x00000002	Message window contains three buttons: Abort, Retry and Ignore
MB_YESNOCANCEL	0x00000003	Message window contains three buttons: Yes, No and Cancel
MB_YESNO	0x00000004	Message window contains two buttons: Yes and No

MB_RETRYCANCEL	0x00000005	Message window contains two buttons: Retry and Cancel
MB_CANCELTRYCONTINUE	0x00000006	Message window contains three buttons: Cancel, Try Again, Continue

To display an icon in the message window it is necessary to specify additional flags:

Constant	Value	Description
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	The STOP sign icon
MB_ICONQUESTION	0x00000020	The question sign icon
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	The exclamation/warning sign icon
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	The encircled i sign

Default buttons are defined by the following flags:

Constant	Value	Description
MB_DEFBUTTON1	0x00000000	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified
MB_DEFBUTTON2	0x00000100	The second button is default
MB_DEFBUTTON3	0x00000200	The third button is default
MB_DEFBUTTON4	0x00000300	The fourth button is default

MQL5 Programs

For the mql5-program to operate, it must be compiled (Compile button or F7 key). Compilation should pass without errors (some warnings are possible; they should be analyzed). At this process, an executable file with the same name and with EX5 extension must be created in the corresponding directory, `terminal_dir\MQL5\Experts`, `terminal_dir\MQL5\indicators` or `terminal_dir\MQL5\scripts`. This file can be run.

Operating features of MQL5 programs are described in the following sections:

- [Program running](#) - order of calling predefined event-handlers.
- [Testing trading strategies](#) - operating features of MQL5 programs in the Strategy Tester.
- [Client terminal events](#) - description of events, which can be processed in programs.
- [Call of imported functions](#) - description order, allowed parameters, search details and call agreement for imported functions.
- [Runtime errors](#) - getting information about runtime and critical errors.

Expert Advisors, custom indicators and scripts are attached to one of opened charts by Drag'n'Drop method from the Navigator window.

For an expert Advisor to stop operating, it should be removed from a chart. To do it select "Expert list" in chart context menu, then select an Expert Advisor from list and click "Remove" button. Operation of Expert Advisors is also affected by the state of the "AutoTrading" button.

In order to stop a custom indicator, it should be removed from a chart.

Custom indicators and Expert Advisors work until they are explicitly removed from a chart; information about attached Expert Advisors and Indicators is saved between client terminal sessions.

Scripts are executed once and are deleted automatically upon operation completion or change of the current chart state, or upon client terminal shutdown. After the restart of the client terminal scripts are not started, because the information about them is not saved.

Maximum one Expert Advisor, one script and unlimited number of indicators can operate in one chart.

Program Running

Each script and each Expert Advisor runs in its own separate thread. All indicators calculated on one symbol, even if they are attached to different charts, work in the same thread. Thus, all indicators on one symbol share the resources of one thread.

All other actions associated with a symbol, like processing of ticks and history synchronization, are also consistently performed in the same thread with indicators. This means that if an infinite action is performed in an indicator, all other events associated with its symbol will never be performed.

When running an Expert Advisor, make sure that it has an actual [trading environment](#) and can [access the history](#) of the required symbol and period, and [synchronize](#) data between the terminal and the server. For all these procedures, the terminal provides a start delay of no more than 5 seconds, after which the Expert Advisor will be started with available data. Therefore, in case there is no connection to the server, this may lead to a delay in the start of an Expert Advisor.

The below table contains a brief summary of MQL5 programs:

Program	Running	Note
Script	A separate thread, the number of threads for scripts is equal to the number of scripts	A looped script cannot break running of other programs
Expert Advisor	A separate thread, the number of threads for Expert Advisors is equal to the number of Expert Advisors	A looped Expert Advisor cannot break running of other programs
Indicator	One thread for all indicators on a symbol. The number of threads is equal to the number of symbols with indicators	An infinite loop in one indicator will stop all other indicators on this symbol

Right after a program is attached to a chart, it is uploaded to the client terminal memory, as well as global variable are [initialized](#). If some global variable of the class type has a [constructor](#), this constructor will be called during initialization of [global variables](#).

After that the program is waiting for an [event](#) from the client terminal. Each mql5-program should have at least one [event-handler](#), otherwise the loaded program will not be executed. Event handlers have predefined names, parameters and return types.

Type	Function name	Parameters	Application	Comment
int	OnInit	none	Expert Advisors and indicators	Init event handler. It allows to use the void return type.
void	OnDeinit	const int reason	Expert Advisors and indicators	Deinit event handler.

void	OnStart	none	scripts	Start event handler.
int	OnCalculate	const int rates_total, const int prev_calculated, const datetime &Time[], const double &Open[], const double &High[], const double &Low[], const double &Close[], const long &TickVolume[], const long &Volume[], const int &Spread[]	indicators	Calculate event handler for all prices.
int	OnCalculate	const int rates_total, const int prev_calculated, const int begin, const double &price[]	indicators	Calculate event handler on the single data array. Indicator cannot have two event handlers simultaneously. In this case the only one event handler will work on the data array.
void	OnTick	none	Expert Advisors	NewTick event handler. While the event of a new tick receipt is being processed, no other events of this type are received.
void	OnTimer	none	Expert Advisors and indicators	Timer event handler.

void	OnTrade	none	Expert Advisors	Trade event handler.
double	OnTester	none	Expert Advisors	Tester event handler.
void	OnChartEvent	const int id, const long &lparam, const double &dparam, const string &sparam	Expert Advisors and indicators	ChartEvent event handler.
void	OnBookEvent	const string &symbol_name	Expert Advisors and indicators	BookEvent event handler.

A client terminal sends new events to the corresponding open charts. Events can also be generated by charts ([chart events](#)) or mql5-programs ([custom events](#)). Generation of events of creation or deletion of graphical objects on a chart can be enabled or disabled by setting [CHART_EVENT_OBJECT_CREATE](#) and [CHART_EVENT_OBJECT_DELETE](#) chart properties. Each MQL5 program and each chart has its own queue of events, where all new incoming events are added.

A program receives only events from the chart it runs on. All events are processed one after another in the order they are received. If a queue already has a [NewTick](#) event, or this event is currently being processed, then the new NewTick event is not placed in the queue of the MQL5 program. Similarly, if [ChartEvent](#) is already enqueued, or this event is being processed, no new event of this kind is enqueued. The timer events are handled the same way - if the [Timer](#) event is in the queue or being handled, the new timer event is not enqueued.

Event queues have a limited but sufficient size, so that the queue overflow for well written programs is unlikely. In case of queue overflow, new events are discarded without queuing.

It is not recommended to use infinite loops to handle events. The exception to this rule may be only scripts that process only a single [Start](#) event.

[Libraries](#) do not handle any events.

Functions prohibited in Indicators and Expert Advisors

Indicators, scripts and Expert Advisors are executable programs written in MQL5. They are designed for different types of tasks. Therefore there are some restrictions on the use of certain functions, depending on the [type of program](#). The following functions are prohibited in indicators:

- [OrderCalcMargin\(\)](#);
- [OrderCalcProfit\(\)](#);
- [OrderCheck\(\)](#);
- [OrderSend\(\)](#);
- [SendFTP\(\)](#);

- [Sleep\(\)](#);
- [ExpertRemove\(\)](#);
- [MessageBox\(\)](#).

All functions designed for indicators are prohibited in Expert Advisors and scripts:

- [SetIndexBuffer\(\)](#);
- [IndicatorSetDouble\(\)](#);
- [IndicatorSetInteger\(\)](#);
- [IndicatorSetString\(\)](#);
- [PlotIndexSetDouble\(\)](#);
- [PlotIndexSetInteger\(\)](#);
- [PlotIndexSetString\(\)](#);
- [PlotIndexGetInteger](#).

The library is not an independent program and is executed in the context of the MQL5 program that has called it: script, indicator or Expert Advisor. Accordingly, the above restrictions apply to the called library.

Loading and Unloading of Indicators

Indicators are loaded in the following cases:

- an indicator is attached to a chart;
- terminal start (if the indicator was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the indicator attached to a chart is specified in the template);
- change of a profile (if the indicator is attached to one of the profile charts);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- after the successful recompilation of an indicator (if the indicator was attached to a chart);
- change of [input parameters](#) of the indicator.

Indicators are unloaded in the following cases:

- when detaching an indicator from a chart;
- terminal shutdown (if the indicator was attached to a chart);
- loading of a template (if an indicator is attached to a chart);
- closing of a chart, to which the indicator was attached;
- change of a profile (if the indicator is attached to one of charts of the changed profile);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- change of [input parameters](#) of the indicator.

Loading and Unloading of Expert Advisors

Expert Advisors are loaded in the following cases:

- when attaching an Expert Advisor to a chart;
- terminal start (if the Expert Advisor was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the Expert Advisor attached to the chart is specified in the template);
- change of a profile (if the Expert Advisor is attached to the one of the profile charts);
- connection to an account, even if the account number is the same (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server).

Expert Advisors are unloaded in the following cases:

- when detaching an Expert Advisor from a chart;
- if a new Expert Advisor is attached to a chart, if another Expert Advisor has been attached already, this Expert Advisor is unloaded.
- terminal shutdown (if the Expert Advisor was attached to a chart);
- loading of a template (if an Expert Advisor is attached to the chart);
- close of a chart, to which the Expert Advisor is attached.
- change of a profile (if the Expert Advisor is attached to one of charts of the changed profile);
- change of the account to which the terminal is connected (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server);
- calling the [ExpertRemove\(\)](#) function.

In case the symbol or timeframe of a chart, to which the Expert Advisor is attached, changes, Expert Advisors are not loaded or unloaded. In this case client terminal subsequently calls [OnDeinit\(\)](#) handlers on the old symbol/timeframe and [OnInit\(\)](#) on the new symbol/timeframe (if they are such), values of global variables and [static variables](#) are not reset. All events, which have been received for the Expert Advisor before the initialization is completed ([OnInit\(\)](#) function) are skipped.

Loading and Unloading of Scripts

Scripts are loaded immediately after they are attached to a chart and unloaded immediately after they complete their operation. [OnInit\(\)](#) and [OnDeinit\(\)](#) are not called for scripts.

When a program is unloaded (deleted from a chart) the client terminal performs deinitialization of [global](#) variables and deletes the events queue. In this case deinitialization means reset of all the [string](#)-type variables, deallocation of [dynamical array objects](#) and call of their [destructors](#) if they are available.

For a better understanding of the Expert Advisor operation we recommend to compile the code of the following Expert Advisor and perform actions of load/unload, template change, symbol change, timeframe change etc:

Example:

```
//+-----+
```



```

//|                                     TestExpert.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
{
public:
    CTestClass() { Print("CTestClass constructor"); }
    ~CTestClass() { Print("CTestClass destructor"); }
};
CTestClass global;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//---
    Print("Initialization");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
    Print("Deinitialization with reason",reason);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//---

}
//+-----+

```

See also

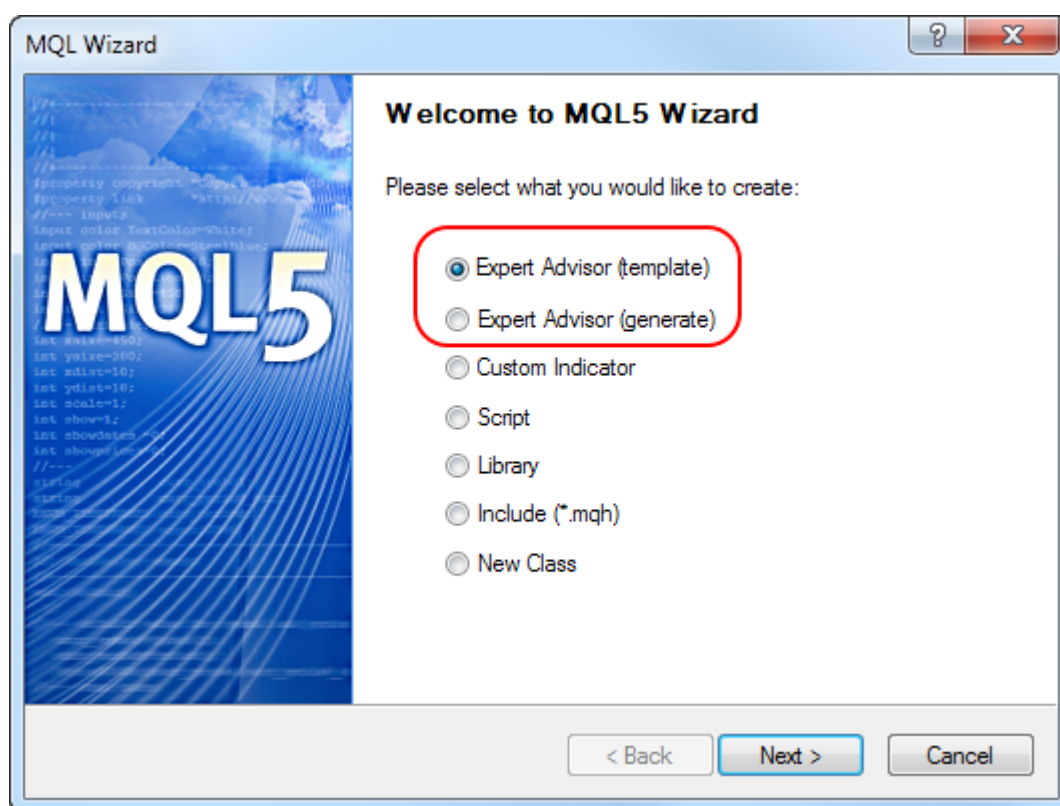
[Client terminal events](#), [Event handlers](#)

Trade Permission

Trade Automation

MQL5 language provides a special group of [trade functions](#) designed for developing automated trading systems. Programs developed for automated trading with no human intervention are called Expert Advisors or trading robots. In order to create an Expert Advisor in MetaEditor, launch MQL5 Wizard and select one of the two options:

- Expert Advisor (template) - allows you to create a template with ready-made [event handling functions](#) that should be supplemented with all necessary functionality by means of programming.
- Expert Advisor (generate) - allows you to [develop a full-fledged trading robot](#) simply by selecting the necessary modules: trading signals module, money management module and trailing stop module.



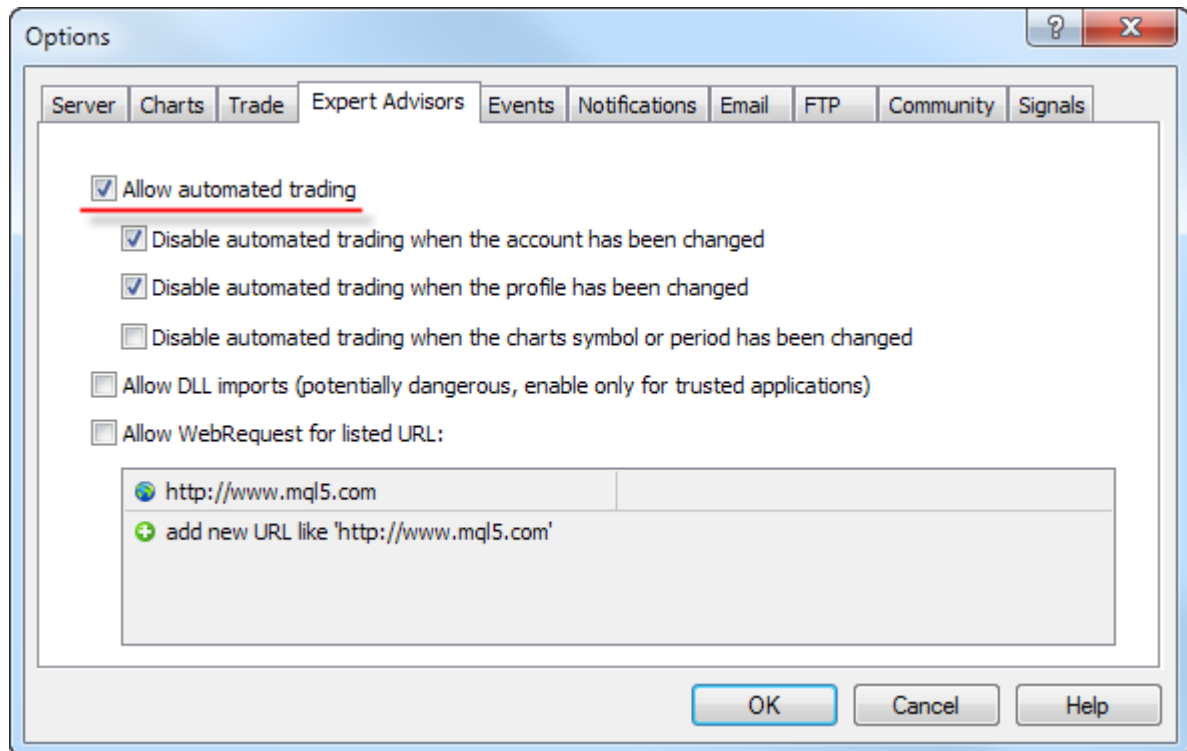
Trading functions can work only in Expert Advisors and scripts. Trading is not allowed for indicators.

Checking for Permission to Perform Automated Trading

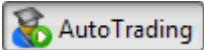
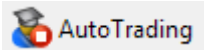
In order to develop a reliable Expert Advisor capable of working without human intervention, it is necessary to arrange a set of important checks. First, we should programmatically check if trading is allowed at all. This is a basic check that is indispensable when developing any automated system.

Checking for permission to perform automated trading in the terminal

The terminal settings provide you with an ability to allow or forbid automated trading for all programs.



You can switch automated trading option right on the terminal's Standard panel:

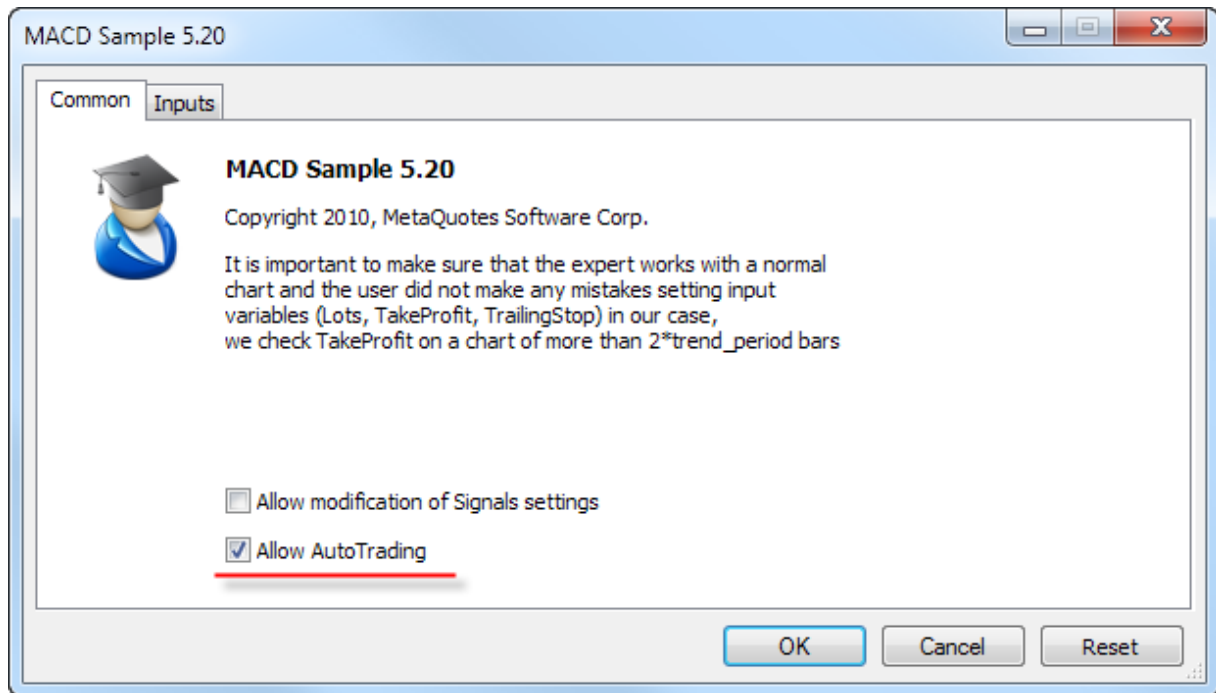
-  - automated trading enabled, trading functions in launched applications are allowed for use.
-  - automated trading disabled, running applications are unable to execute trading functions.

Sample check:

```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
    Alert("Check if automated trading is allowed in the terminal settings!");
```

Checking if trading is allowed for a certain running Expert Advisor/script

You can allow or forbid automated trading for a certain program when launching it. To do this, use the special check box in the program properties.



Sample check:

```
if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
    Alert("Check if automated trading is allowed in the terminal settings!");
else
{
    if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
        Alert("Automated trading is forbidden in the program settings for ",__FILE__)
}
```

Checking if trading is allowed for any Expert Advisors/scripts for the current account

Automated trading can be disabled at the trade server side. Sample check:

```
if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT))
    Alert("Automated trading is forbidden for the account ",AccountInfoInteger(ACCOUNT_LOGIN)
    " at the trade server side");
```

If automated trading is disabled for a trading account, trading operations of Expert Advisors/scripts are not executed.

Checking if trading is allowed for the current account

In some cases, any trading operations are disabled for a certain trading account - neither manual nor automated trading can be performed. Sample check when an investor password has been used to connect to a trading account:

```
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
    Comment("Trading is forbidden for the account ",AccountInfoInteger(ACCOUNT_LOGIN)
```



```
".\n Perhaps an investor password has been used to connect to the trading
"\n Check the terminal journal for the following entry:",
"\n\'",AccountInfoInteger(ACCOUNT_LOGIN),"\': trading has been disabled -
```

AccountInfoInteger(ACCOUNT_TRADE_ALLOWED) may return **false** in the following cases:

- no connection to the trade server. That can be checked using TerminalInfoInteger(TERMINAL_CONNECTED);
- trading account switched to read-only mode (sent to the archive);
- trading on the account is disabled at the trade server side;
- connection to a trading account has been performed in Investor mode.

See also

[Client Terminal Properties](#), [Account Properties](#), [Properties of a Running MQL5 Program](#)

Client Terminal Events

Init

Immediately after the client terminal loads a program (an Expert Advisor or custom indicator) and starts the process of initialization of global variables, the Init event will be sent, which will be processed by [OnInit\(\)](#) event handler, if there is such. This event is also generated after a financial instrument and/or chart timeframe is changed, after a program is recompiled in MetaEditor, after input parameters are changed from the setup window of an Expert Advisor or a custom indicator. An Expert Advisor is also initialized after the account is changed. The Init event is not generated for scripts.

Deinit

Before global variables are deinitialized and the program (Expert Advisor or custom indicator) is unloaded, the client terminal sends the Deinit event to the program. Deinit is also generated when the client terminal is closed, when a chart is closed, right before the security and/or timeframe is changed, at a successful program re-compilation, when input parameters are changed, and when account is changed.

The [deinitialization reason](#) can be obtained from the parameter, passed to the [OnDeinit\(\)](#) function. The OnDeinit() function run is restricted to 2.5 seconds. If during this time the function hasn't been completed, then it is forcibly terminated. The Deinit event is not generated for scripts.

Start

The [Start](#) event is a special event for script activation after it is loaded. This event is processed by [OnStart](#) handler. The Start event is not sent to Expert Advisors or custom indicators.

NewTick

The [NewTick](#) event is generated if there are new quotes, it is processed by [OnTick\(\)](#) of Expert Advisors attached. In case when OnTick function for the previous quote is being processed when a new quote is received, the new quote will be ignored by an Expert Advisor, because the corresponding event will not enqueued.

All new quotes that are received while the program is running are ignored until the OnTick() is completed. After that the function will run only after a new quote is received. The NewTick event is generated irrespective of whether automated trade is allowed or not ("Allow/prohibit Auto trading" button). The prohibition of automated trading denotes only that sending of trade requests from an Expert Advisor is not allowed, while the Expert Advisor keeps working.

The prohibition of automated trading by pressing the appropriate button will not stop the current execution of the OnTick() function.

Calculate

The [Calculate](#) event is generated only for indicators right after the Init event is sent and at any change of price data. It is processed by the [OnCalculate](#) function.

Timer

The **Timer** event is periodically generated by the client terminal for the Expert Advisor that has activated the timer by the [EventSetTimer](#) function. Usually, this function is called by OnInit. Timer event processing is performed by the [OnTimer](#) function. After the operation of the Expert Advisor is completed, it is necessary to destroy the timer using the [EventKillTimer](#) function, which is usually called in the OnDeinit function.

Trade

The Trade event is generated when a trade operation is completed on a trade server. The Trade event is handled by the [OnTrade\(\)](#) function for the following trade operations:

- sending, modifying or removing of a pending order;
- cancellation of a pending order with not enough of money or expiration;
- activation of a pending order;
- opening, adding or closing a position (or part of the position);
- modifying of the open position (change stops - Stop Loss and/or Take Profit).

TradeTransaction

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using [OrderSend](#) and [OrderSendAsync](#) functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions. Arrival of such a transaction at the terminal is a TradeTransaction event. This event is handled by [OnTradeTransaction](#) function.

Tester

The **Tester** event is generated after testing of an Expert Advisor on history data is over. The event is handled by the [OnTester\(\)](#) function.

TesterInit

The [TesterInit](#) event is generated with the start of optimization in the strategy tester before the first optimization pass. The TesterInit event is handled by the [OnTesterInit\(\)](#) function.

TesterPass

The [TesterPass](#) event is generated when a new [data frame](#) is received. The TesterPass event is handled by the [OnTesterPass\(\)](#) function.

TesterDeinit

The [TesterDeinit](#) event is generated after the end of optimization of an Expert Advisor in the strategy tester. The TesterDeinit event is handled by the [OnTesterDeinit\(\)](#) function.

ChartEvent

The [ChartEvent](#) event is generated by the client terminal when a user is working with a chart:

- keystroke, when the chart window is in focus;
- [graphical object](#) created
- [graphical object](#) deleted
- mouse press on the graphical object of the chart
- move of the graphical object using the mouse
- end of text editing in LabelEdit.

Also there is a custom event [ChartEvent](#), which can be sent to an Expert Advisor by any mql5 program by using the [EventChartCustom](#) function. The event is processed by the [OnChartEvent](#) function.

BookEvent

The [BookEvent](#) event is generated by the client terminal after the Depth Of Market is changed; it is processed by the [OnBookEvent](#) function. To start generation of BookEvent for the specified symbol, it is necessary to subscribe the symbol to this event by using the [MarketBookAdd](#) function.

To unsubscribe from BookEvent for a specified symbol, it is necessary to call the [MarketBookRelease](#) function. The BookEvent event is a broadcasting-type event - it means that it is sufficient to subscribe just one Expert Advisor for this event, and all other Expert Advisors that have the OnBookEvent event handler, will receive it. That's why it is necessary to analyze the symbol name, which is passed to a handler as a parameter.

See also

[Event handlers](#), [Program running](#)

Resources

Using graphics and sound in MQL5 programs

Programs in MQL5 allow working with sound and graphic files:

- [PlaySound\(\)](#) plays a sound file;
- [ObjectCreate\(\)](#) allows creating user interfaces using [graphical objects](#) OBJ_BITMAP and OBJ_BITMAP_LABEL.

PlaySound()

Example of call of the [PlaySound\(\)](#) function:

```
//+-----+
//| Calls standard OrderSend() and plays a sound |
//+-----+
void OrderSendWithAudio(MqlTradeRequest &request, MqlTradeResult &result)
{
    //--- send a request to a server
    OrderSend(request,result);
    //--- if a request is accepted, play sound Ok.wav
    if(result.retcode==TRADE_RETCODE_PLACED) PlaySound("Ok.wav");
    //--- if fails, play alarm from file timeout.wav
    else PlaySound("timeout.wav");
}
```

The example shows how to play sounds from files 'Ok.wav' and 'timeout.wav', which are included into the standard terminal package. These files are located in the folder **terminal_directory\Sounds**. Here **terminal_directory** is a folder, from which the MetaTrader 5 Client Terminal is started. The location of the terminal directory can be found out from an mql5 program in the following way:

```
//--- Folder, in which terminal data are stored
string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

You can use sound files not only from the folder **terminal_directory\Sounds**, but also from any subfolder located in **terminal_data_directory\MQL5**. You can find out the location of the terminal data directory from the terminal menu "File" -> "Open Data Folder" or using program method:

```
//--- Folder, in which terminal data are stored
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

For example, if the Demo.wav sound file is located in **terminal_data_directory\MQL5\Files**, then call of **PlaySound()** should be written the following way:

```
//--- play Demo.wav from the folder terminal_directory_data\MQL5\Files\
PlaySound("\\Files\\Demo.wav");
```

Please note that in the comment the path to the file is written using backslash "\", and in the function "\\\" is used.

When specifying the path, always use only the double backslash as a separator, because a single backslash is a control symbol for the compiler when dealing with constant strings and [character constants](#) in the program source code.

Call [PlaySound\(\)](#) function with NULL parameter to stop playback:

```
//--- call of PlaySound() with NULL parameter stops playback
PlaySound(NULL);
```

ObjectCreate()

Example of an Expert Advisor, which creates a graphical label (OBJ_BITMAP_LABEL) using the ObjectCreate() function.

```
string label_name="currency_label";      // name of the OBJ_BITMAP_LABEL object
string euro      ="\\Images\\euro.bmp";  // path to the file terminal_data_directory
string dollar    ="\\Images\\dollar.bmp"; // path to the file terminal_data_directory
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- create a button OBJ_BITMAP_LABEL, if it hasn't been created yet
if(ObjectFind(0,label_name)<0)
{
//--- trying to create object OBJ_BITMAP_LABEL
bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
if(created)
{
//--- link the button to the left upper corner of the chart
ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
//--- now set up the object properties
ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
//--- reset the code of the last error to 0
ResetLastError();
//--- download a picture to indicate the "Pressed" state of the button
bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
//--- test the result
if(!set)
{
PrintFormat("Failed to download image from file %s. Error code %d",euro,GetLastError());
ResetLastError();
//--- download a picture to indicate the "Unpressed" state of the button
set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

if(!set)
```



```

        {
            PrintFormat("Failed to download image from file %s. Error code %d",dollar,
        }
        //-- send a command for a chart to refresh so that the button appears immedi
        ChartRedraw(0);
    }
    else
    {
        //-- failed to create an object, notify
        PrintFormat("Failed to create object OBJ_BITMAP_LABEL. Error code %d",GetLast
    }
}
//--
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //-- delete an object from a chart
    ObjectDelete(0,label_name);
}

```

Creation and setup of the graphical object named `currency_label` are carried out in the `OnInit()` function. The paths to the graphical files are set in [global variables](#) `euro` and `dollar`, a double backslash is used for a separator:

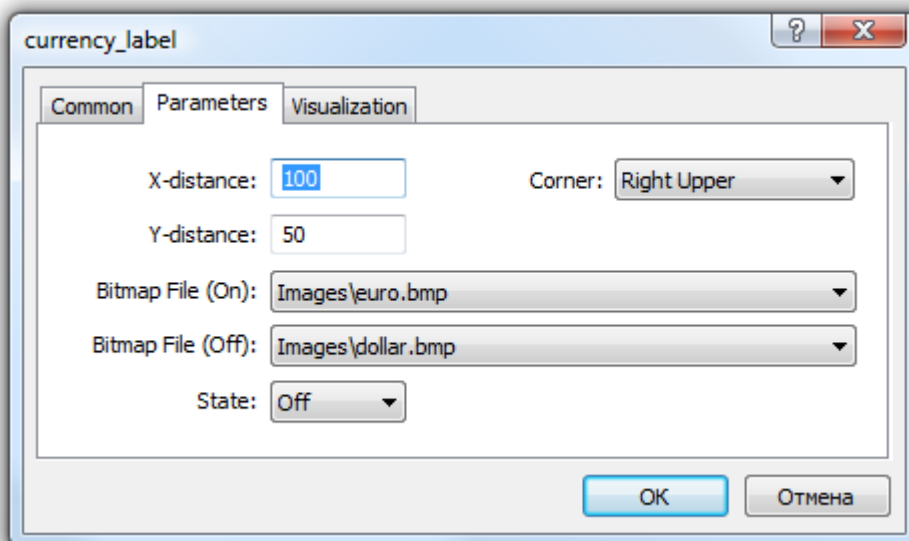
```

string euro      ="\\Images\\euro.bmp";    // path to the file terminal_data_directory
string dollar    ="\\Images\\dollar.bmp";  // path to the file terminal_data_directory

```

The files are located in the folder `terminal_data_directory\MQL5\Images`.

Object `OBJ_BITMAP_LABEL` is actually a button, which displays one of the two images, depending on the button state (pressed or unpressed): `euro.bmp` or `dollar.bmp`.



The size of the button with a graphical interface is automatically adjusted to the size of the picture. The image is changed by a left mouse button click on the OBJ_BITMAP_LABEL object ("Disable selection" option must be checked in the properties). The OBJ_BITMAP object is created the same way - it is used for creating the background with a necessary image.

The value of the [OBJPROP_BMPFILE](#) property, which is responsible for the appearance of the objects OBJ_BITMAP and OBJ_BITMAP_LABEL, can be changed dynamically. This allows creating various interactive user interfaces for mql5 programs.

Including resources to executable files during compilation of mql5 programs

An mql5 program may need a lot of different downloadable resources in the form of image and sound files. In order to eliminate the need to transfer all these files when moving an executable file in MQL5, the compiler's directive `#resource` should be used:

```
#resource path_to_resource_file
```

The `#resource` command tells the compiler that the resource at the specified path `path_to_resource_file` should be included into the executable EX5 file. Thus all the necessary images and sounds can be located directly in an EX5 file, so that there is no need to transfer separately the files used in it, if you want to run the program on a different terminal. Any EX5 file can contain resources, and any EX5 program can use resources from another EX5 program.

The files in format BMP and WAV are automatically compressed before including them to an EX5 file. This denotes that in addition to the creation of complete programs in MQL5, using resources also allows to reduce the total size of necessary files when using graphics and sounds, as compared to the usual way of MQL5 program writing.

The resource file size must not exceed 128 Mb.

Search for specified resources by a compiler

A resource is inserted using the command `#resource "<path to the resource file>"`

```
#resource "<path_to_resource_file>"
```

The length of the constant string `<path_to_resource_file>` must not exceed 63 characters.

The compiler searches for a resource at the specified path in the following order:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, it searches for the resource relative to the directory `terminal_data_directory\MQL5\`,
- if there is no backslash, it searches for the resource relative to the location of the source file, in which the resource is written.

The resource path cannot contain the substrings `".\\`" and `":\\`".

Examples of resource inclusion:

```
//--- correct specification of resources
#resource "\\Images\\euro.bmp" // euro.bmp is located in terminal_data_directory\MQL5\
#resource "picture.bmp"        // picture.bmp is located in the same directory as the
#resource "Resource\\map.bmp"  // the resource is located in source_file_directory\Res

//--- incorrect specification of resources
#resource ":picture_2.bmp"      // must not contain ":"
#resource "..\\picture_3.bmp"  // must not contain ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.bmp" //more
```

Use of Resources

Resource name

After a resource is declared using the `#resource` directive, it can be used in any part of a program. The name of the resource is its path without a backslash at the beginning of the line, which sets the path to the resource. To use your own resource in the code, the special sign `::` should be added before the resource name.

Examples:

```
//--- examples of resource specification and their names in comments
#resource "\\Images\\euro.bmp"           // resource name - Images\euro.bmp
#resource "picture.bmp"                  // resource name - picture.bmp
#resource "Resource\\map.bmp"            // resource name - Resource\map.bmp
#resource "\\Files\\Pictures\\good.bmp"  // resource name - Files\Pictures\good.bmp
#resource "\\Files\\Demo.wav";           // resource name - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";        // resource name - Sounds\thrill.wav"
...

//--- utilization of resources
```

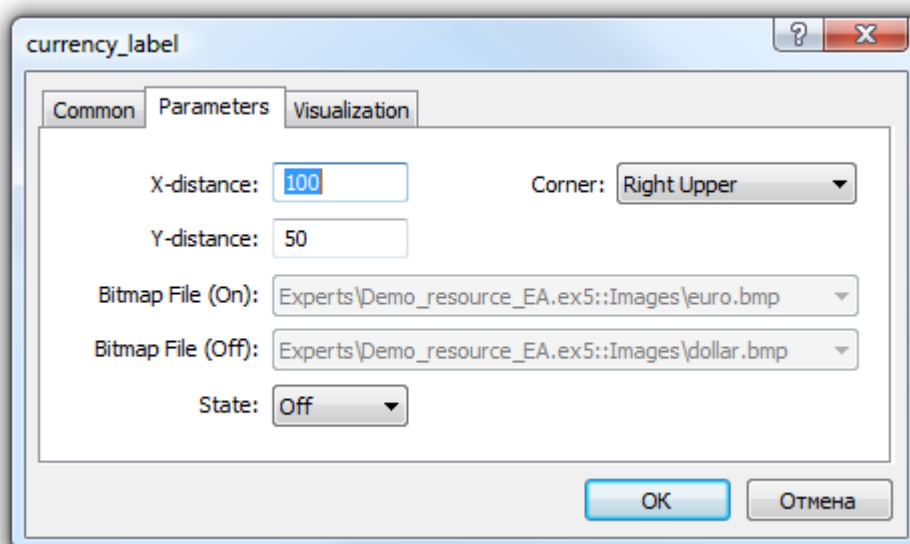


```
ObjectSetString(0,bitmap_name,OBJPROP_BMPFILE,0,"::Images\\euro.bmp");
...
ObjectSetString(0,my_bitmap,OBJPROP_BMPFILE,0,"::picture.bmp");
...
set=ObjectSetString(0,bitmap_label,OBJPROP_BMPFILE,1,"::Files\\Pictures\\good.bmp");
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

It should be noted that when setting images from a resource to the OBJ_BITMAP and OBJ_BITMAP_LABEL objects, the value of the OBJPROP_BMPFILE property cannot be modified manually. For example, for creating OBJ_BITMAP_LABEL we use resources euro.bmp and dollar.bmp.

```
#resource "\\Images\\euro.bmp"; // euro.bmp is located in terminal_data_directory\
#resource "\\Images\\dollar.bmp"; // dollar.bmp is located in terminal_data_directory\
```

When viewing the properties of this object, we'll see that the properties BitMap File (On) and BitMap File (Off) are dimmed and cannot be change manually:



Using the resources of other mql5 programs

There is another advantage of using resources - in any MQL5 program, resources of another EX5 file can be used. Thus the resources from one EX5 file can be used in many other mql5 programs.

In order to use a resource name from another file, it should be specified as <path_EX5_file_name>::<resource_name>. For example, suppose the Draw_Triangles_Script.mq5 script contains a resource to an image in the file triangle.bmp:

```
#resource "\\Files\\triangle.bmp"
```

Then its name, for using in the script itself, will look like "Files\\triangle.bmp", and in order to use it, "::" should be added to the resource name.


```
//--- using the resource in the script
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp");
```

In order to use the same resource from another program, e.g. from an Expert Advisor, we need to add to the resource name the path to the EX5 file relative to **terminal_data_directory\MQL5** and the name of the script's EX5 file - **Draw_Triangles_Script.ex5**. Suppose the script is located in the standard folder **terminal_data_directory\MQL5\Scripts**, then the call should be written the following way:

```
//--- using a resource from a script in an EA
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triangles_Script.e
```

If the path to the executable file is not specified when calling the resource from another EX5, the executable file is searched for in the same folder that contains the program that calls the resource. This means that if an Expert Advisor calls a resource from **Draw_Triangles_Script.ex5** without specification of the path, like this:

```
//--- call script resource in an EA without specifying the path
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.ex5::Files\\
```

then the file will be searched for in the folder **terminal_data_directory\MQL5\Experts**, if the Expert Advisor is located in **terminal_data_directory\MQL5\Experts**.

Working with custom indicators included as resources

One or several custom indicators may be necessary for the operation of MQL5 applications. All of them can be included into the code of an executable MQL5 program. Inclusion of indicators as resources simplifies the distribution of applications.

Below is an example of including and using **SampleIndicator.ex5** custom indicator located in **terminal_data_folder\MQL5\Indicators** directory:

```
//+-----+
//|                                     SampleEA.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#resource "\\Indicators\\SampleIndicator.ex5"
int handle_ind;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//---
    handle_ind=iCustom(_Symbol,_Period,"::Indicators\\SampleIndicator.ex5");
    if(handle_ind==INVALID_HANDLE)
    {
        Print("Expert: iCustom call: Error code=",GetLastError());
        return(INIT_FAILED);
    }
}
```



```

    }
//--- ...
    return(INIT_SUCCEEDED);
}

```

The case when a custom indicator in [OnInit\(\)](#) function creates one or more copies of itself requires special consideration. Please keep in mind that the resource should be specified in the following way: `<path_EX5_file_name>::<resource_name>`.

For example, if SampleIndicator.ex5 indicator is included to SampleEA.ex5 Expert Advisor as a resource, the path to itself specified when calling [iCustom\(\)](#) in the custom indicator's initialization function looks the following way: `"\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5"`. When this path is set explicitly, SampleIndicator.ex5 custom indicator is rigidly connected to SampleEA.ex5 Expert Advisor losing ability to work independently.

The path to itself can be received using `GetRelativeProgramPath()` function. The example of its usage is provided below:

```

//+-----+
//|                                     SampleIndicator.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property indicator_separate_window
#property indicator_plots 0
int handle;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- the wrong way to provide a link to itself
//--- string path="\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5";
//--- the right way to receive a link to itself
    string path=GetRelativeProgramPath();
//--- indicator buffers mapping
    handle=iCustom(_Symbol,_Period,path,0,0);
    if(handle==INVALID_HANDLE)
    {
        Print("Indicator: iCustom call: Error code=",GetLastError());
        return(INIT_FAILED);
    }
    else Print("Indicator handle=",handle);
//---
    return(INIT_SUCCEEDED);
}
//....
//+-----+
//| GetRelativeProgramPath |
//+-----+

```



```

string GetRelativeProgramPath()
{
    int pos2;
    ///--- get the absolute path to the application
    string path=MQLInfoString(MQL_PROGRAM_PATH);
    ///--- find the position of "\\MQL5\\" substring
    int pos =StringFind(path,"\\MQL5\\");
    ///--- substring not found - error
    if(pos<0)
        return(NULL);
    ///--- skip "\\MQL5" directory
    pos+=5;
    ///--- skip extra '\\' symbols
    while(StringGetCharacter(path,pos+1)=='\\')
        pos++;
    ///--- if this is a resource, return the path relative to MQL5 directory
    if(StringFind(path,":",pos)>=0)
        return(StringSubstr(path,pos));
    ///--- find a separator for the first MQL5 subdirectory (for example, MQL5\\Indicators)
    ///--- if not found, return the path relative to MQL5 directory
    if((pos2=StringFind(path,"\\",pos+1))<0)
        return(StringSubstr(path,pos));
    ///--- return the path relative to the subdirectory (for example, MQL5\\Indicators)
    return(StringSubstr(path,pos2+1));
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const int begin,
               const double& price[])
{
    ///--- return value of prev_calculated for next call
    return(rates_total);
}

```

Resource variables

Resources can be declared using the resource variables and treated as if they are variables of the appropriate type. Declaration format:

```
#resource path_to_the_resource_file as resource_variable_type resource_variable_name
```

Sample declarations:

```
#resource "data.bin" as int ExtData[] // declare the numeric array contain
```

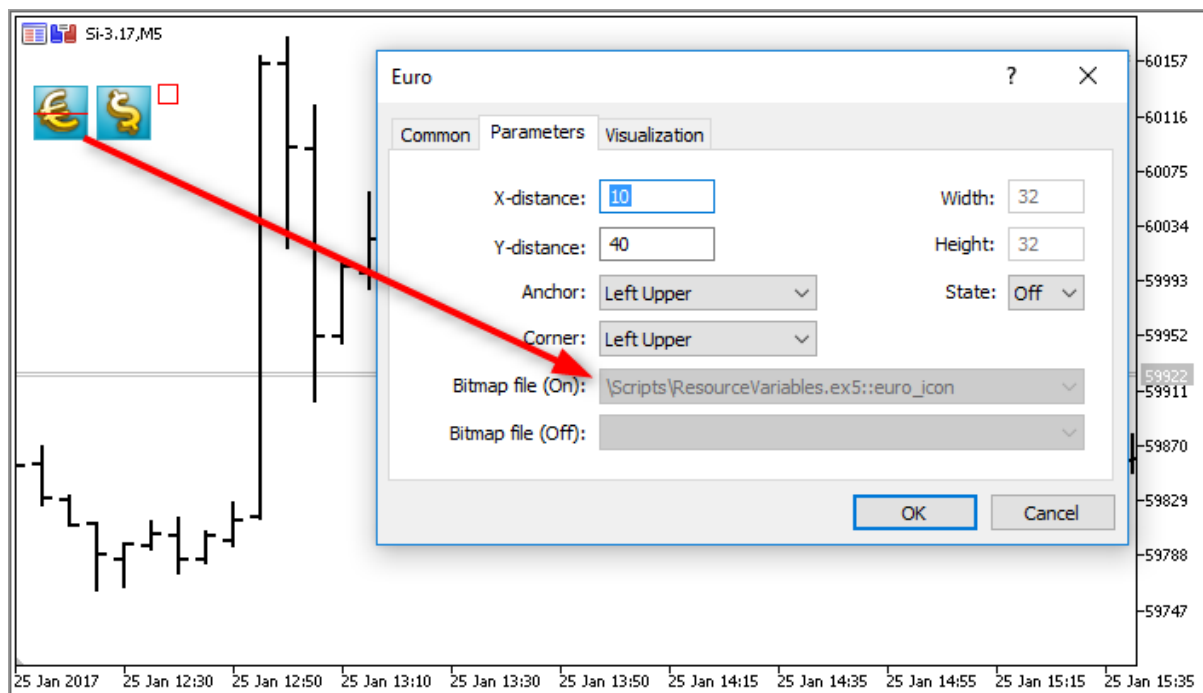


```
#resource "data.bin" as MqlRates ExtData[] // declare the simple structures array
//--- strings
#resource "data.txt" as string ExtCode // declare the string containing the
//--- graphical resources
#resource "image.bmp" as bitmap ExtBitmap[] // declare the one-dimensional array
#resource "image.bmp" as bitmap ExtBitmap2[][] // declare the two-dimensional array
```

In case of such declaration, the resource data can be addressed only via the variable, **auto addressing** via `::<resource name>` **does not work**.

```
#resource "\\Images\\euro.bmp" as bitmap euro[][]
#resource "\\Images\\dollar.bmp"
//+-----+
//| OBJ_BITMAP_LABEL object creation function using the resource |
//+-----+
void Image(string name,string rc,int x,int y)
{
    ObjectCreate(0,name,OBJ_BITMAP_LABEL,0,0,0);
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
    ObjectSetString(0,name,OBJPROP_BMPFILE,rc);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- output the size of the image [width, height] stored in euro resource variable
    Print(ArrayRange(euro,1)," ",ArrayRange(euro,0));
    //--- change the image in euro - draw the red horizontal stripe in the middle
    for(int x=0;x<ArrayRange(euro,1);x++)
        euro[ArrayRange(euro,1)/2][x]=0xFFFF0000;
    //--- create the graphical resource using the resource variable
    ResourceCreate("euro_icon",euro,ArrayRange(euro,0),ArrayRange(euro,1),0,0,ArrayRange(euro,1));
    //--- create the Euro graphical label object, to which the image from the euro_icon resource is assigned
    Image("Euro","::euro_icon",10,40);
    //--- another method of applying the resource, we cannot draw do it
    Image("USD","::Images\\dollar.bmp",15+ArrayRange(euro,1),40);
    //--- direct method of addressing the euro.bmp resource is unavailable since it has a
    Image("E2","::Images\\euro.bmp",20+ArrayRange(euro,1)*2,40); // execution time error
}
```

Script execution result - only two OBJ_BITMAP_LABEL objects out of three ones are created. The image of the first object has the red stripe in the middle.



An important advantage of applying the resources is that the resource files are automatically compressed before they are included into an executable EX5 file prior to compilation. Thus, using the resource variables allows you to put all necessary data directly into the executable EX5 file as well as reduce the number and total size of the files compared to the conventional way of writing MQL5 programs.

Using the resource variables is particularly convenient for publishing products in the [Market](#).

Features

- The special *bitmap* resource variable type informs the compiler that the resource is an image. Such variables receive the uint type.
- The *bitmap* type array resource variable may have two dimensions. In this case, the array size is defined as [image_height][image_width]. If an array of one dimension is specified, the number of elements is equal to image_height*image_width.
- When downloading a 24-bit image, the [alpha channel](#) component is set to 255 for all the image pixels.
- When downloading a 32-bit image without the alpha channel, the alpha channel component is also set to 255 for all the image pixels.
- When downloading a 32-bit image with the alpha channel, the pixels are not processed in any way.
- The resource file size cannot exceed 128 Mb.
- The automatic encoding detection by BOM (header) presence is performed for string files. If BOM is absent, the encoding is defined by the file contents. The files in the ANSI, UTF-8 and UTF-16 encodings are supported. All strings are converted to Unicode when reading data from the files.

OpenCL programs

Using the resource string variables may greatly facilitate the development of some programs. For example, you are able to write a code of an [OpenCL program](#) in a separate CL file and then include it as a string into your MQL5 program resources.


```
#resource "seascape.cl" as string cl_program
...
int context;
if((cl_program=CLProgramCreate(context,cl_program)!=INVALID_HANDLE)
{
    //-- perform further actions with an OpenCL program
}
```

In this example, you would have had to write the entire code as a single big string if no *cl_program* resource variable had been used.

See also

[ResourceCreate\(\)](#), [ResourceSave\(\)](#), [PlaySound\(\)](#), [ObjectSetInteger\(\)](#), [ChartApplyTemplate\(\)](#), [File Functions](#)

Call of Imported Functions

To import functions during the execution of a mql5-program, the client terminal uses early binding. This means that if a program has call of an imported function, the corresponding module (ex5 or dll) is loaded during the program load. MQL5 and DLL libraries are executed in the thread of a calling module.

It is not recommended to use the fully specified name of the module to be loaded like *Drive:\Directory\FileName.Ext*. The MQL5 libraries are loaded from the *terminal_dir\MQL5\Libraries* folder. If the library hasn't been found, then the client terminal performs an attempt to load it from *terminal_dir\experts* folder.

The system libraries (DLL) are loaded by the operating system rules. If the library is already loaded (for example, another Expert Advisor, and even from another client terminal, running in parallel), then it uses requests to the library already loaded. Otherwise, it performs a search in the following sequence:

1. Directory, from which the module importing dll was started. The module here is an Expert Advisor, a script, an indicator or EX5 library;
2. Directory *terminal_data_directory\MQL5\Libraries* ([TERMINAL_DATA_PATH\MQL5\Libraries](#));
3. Directory, from which the MetaTrader 5 client terminal was started;
4. System directory;
5. Windows directory;
6. Current directory;
7. Directories listed in the PATH system variable.

If the DLL library uses another DLL in its work, the first one cannot be loaded in case when there is no second DLL.

Before an Expert Advisor (script, indicator) is loaded, a common list of all EX5 library modules is formed. It's going to be used both from a loaded Expert Advisor (script, indicator) and from libraries of this list. Thus the one-time loading of many times used EX5 library modules is needed. Libraries use [predefined variables](#) of the Expert Advisor (script, indicator) they were called by.

The imported library EX5 is searched for in the following sequence:

1. Directory, path to which is set relative to the directory of the Expert Advisor (script, indicator) that imports EX5;
2. Directory *terminal_directory\MQL5\Libraries*;
3. Directory *MQL5\Libraries* in the common directory of all MetaTrader 5 client terminals (*Common\MQL5\Libraries*).

Functions [imported](#) DLL into a mql5-program must ensure the Windows API calls agreement. To ensure such an agreement, in the source text of programs written in C or C++, use the keyword `__stdcall`, which is specific to the Microsoft(r) compilers. This agreement is characterized by the following:

- caller (in our case it is a mql5-program) should "see" a prototype of a function called (imported from the DLL), in order to properly combine parameters to a stack;
- caller (in our case it is a mql5-program) puts parameters to the stack in a reverse order, from right to left - in this order an imported function reads parameters passed to it;
- parameters are passed by value, except those explicitly passed by reference (in our case strings)

- an imported function cleans the stack independently by reading parameters passed to it.

When describing the prototype of an imported function, default parameters can be used.

If the corresponding library is unable to load, or there is a prohibition on the DLL use, or the imported function is not found - the Expert Advisor stops its operation with the appropriate message "Expert Advisor stopped" in the Journal (log file). In this case the Expert Advisor will not run until it is reinitialized. An Expert Advisor can be reinitialized as a result of recompilation or after the table of its properties is opened and OK is pressed.

Passing Parameters

All parameters of [simple types](#) are passed by values unless it is explicitly indicated that they are passed by reference. When a [string](#) is passed, the address of the buffer of the copied string is passed; if a string is passed by reference, the address of the buffer of this string without copying it is passed to the function imported from DLL.

[Structures](#) that contain dynamic arrays, strings, classes, other complex structures, as well as static or [dynamic arrays](#) of the enumerated objects, can't be passed as a parameter to an imported function.

When passing an array to DLL, the address of the beginning of the data buffer is always passed (irrespective of the [AS_SERIES](#) flag). A function inside a DLL knows nothing about the AS_SERIES flag, the passed array is a static array of an undefined length; an additional parameter should be used for specifying the array size.

Runtime Errors

The executing subsystem of the client terminal has an opportunity to save the [error code](#) in case it occurs during a MQL5 program run. There is a predefined variable [_LastError](#) for each executable MQL5 program.

Before starting the [OnInit](#) function, the `_LastError` variable is reset. In case an erroneous situation occurs during calculations or in the process of internal function calls, the `_LastError` variable accepts a corresponding error code. The value stored in this variable can be obtained using the [GetLastError\(\)](#) function.

There are several critical errors in case of which a program is terminated immediately:

- division by zero
- going beyond array boundary
- using an incorrect [object pointer](#)

Testing Trading Strategies

The idea of automated trading is appealing by the fact that the trading robot can work non-stop for 24 hours a day, seven days a week. The robot does not get tired, doubtful or scared, it's is totally free from any psychological problems. It is sufficient enough to clearly formalize the trading rules and implement them in the algorithms, and the robot is ready to work tirelessly. But first, you must make sure that the following two important conditions are met:

- The Expert Advisor performs [trading operations](#) in accordance with the rules of the trading system;
- The trading strategy, implemented in the EA, demonstrates a profit on the history.

To get answers to these questions, we turn to the [Strategy Tester](#), included in the MetaTrader 5 client terminal.

This section covers the features of program testing and optimization in the strategy tester:

- [Function Limitations in the Strategy Tester](#)
- [Tick Generation Modes](#)
- [Simulation of spread](#)
- [Using real ticks during a test](#)
- [The Global Variables of the Client Terminal](#)
- [The Calculation of Indicators During Testing](#)
- [Loading History during Testing](#)
- [Multi-Currency Testing](#)
- [Simulation of Time in the Strategy Tester](#)
- [Graphical Objects in Testing](#)
- [The OnTimer\(\) Function in the Strategy Tester](#)
- [The Sleep\(\) Function in the Strategy Tester](#)
- [Using the Strategy Tester for Optimization Problems in Mathematical Calculations](#)
- [The Synchronization of Bars in the "Open prices only" mode](#)
- [The IndicatorRelease\(\) function in the Tester](#)
- [Event Handling in the Tester](#)
- [Testing Agents](#)
- [The Data Exchange between the Terminal and the Agent](#)
- [Using the Shared Folder of All of the Client Terminals](#)
- [Using DLLs](#)

Function Limitations in the Strategy Tester

There are operation limitations for some functions in the client terminal's Strategy Tester.

The Print() and PrintFormat() Functions

To increase performance, [Print\(\)](#) and [PrintFormat\(\)](#) functions are not executed when optimizing the trading robot's parameters. The exception is the use of these functions inside the [OnInit\(\)](#) handler. This allows you to easily find the cause of errors when they occur.

The [Alert\(\)](#), [MessageBox\(\)](#), [PlaySound\(\)](#), [SendFTP\(\)](#), [SendMail\(\)](#), [SendNotification\(\)](#), [WebRequest\(\)](#) Functions

The [Alert\(\)](#), [MessageBox\(\)](#), [PlaySound\(\)](#), [SendFTP\(\)](#), [SendMail\(\)](#), [SendNotification\(\)](#) and [WebRequest\(\)](#) functions designed for interaction with the "outside world" are not executed in the Strategy Tester.

Tick Generation Modes

An Expert Advisor is a program, written in MQL5, that is run each time in response to some external [event](#). The EA has a corresponding function ([event handler](#)) for each pre-defined event.

The [NewTick](#) event (price change) is the main event for the EA and, therefore, we need to generate a tick sequence to test the EA. There are 3 modes of tick generation implemented in the Strategy Tester of MetaTrader 5 client terminal:

- Every tick
- 1 Minute OHLC (OHLC prices with minute bars)
- Open prices only

The basic and the most detailed is the "Every tick" mode, the other two modes are the simplifications of the basic one, and will be described in comparison to the "Every tick" mode. Consider all three modes in order to understand the differences between them.

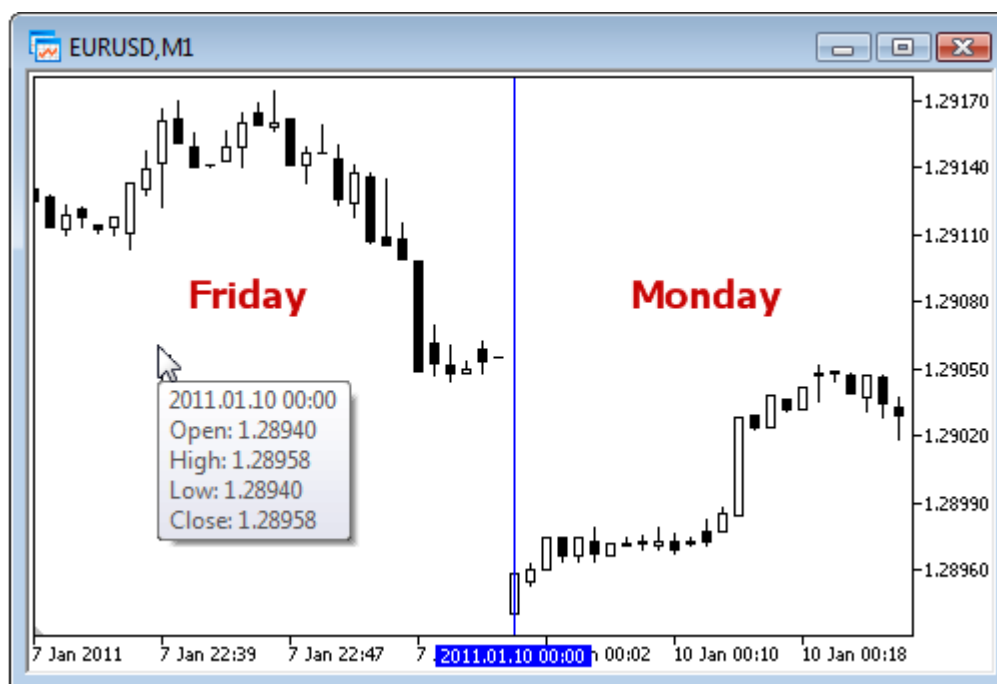
"Every Tick"

The historical quotes data for financial instruments is transferred from the trading server to the MetaTrader 5 client terminal in the form of packed minute bars. Detailed information on the occurrence of requests and the construction of the required time-frames can be obtained from the [Organizing Data Access](#) chapter of MQL5 Reference.

The minimal element of the price history is the minute bar, from which you can obtain information on the four values of the price:

- Open - the price at which the minute bar was opened;
- High - the maximum that was achieved during this minute bar;
- Low - the minimum that was achieved during this minute bar;
- Close - the closing price of the bar.

The new minute bar is not opened at the moment when the new minute begins (number of seconds becomes equal to 0), but when a tick occurs - a price change by at least one point. The figure shows the first minute bar of the new trading week, which has the opening time of 2011.01.10 00:00. The price gap between Friday and Monday, which we see on the chart, is common, since currency rates fluctuates even on weekends in response to incoming news.

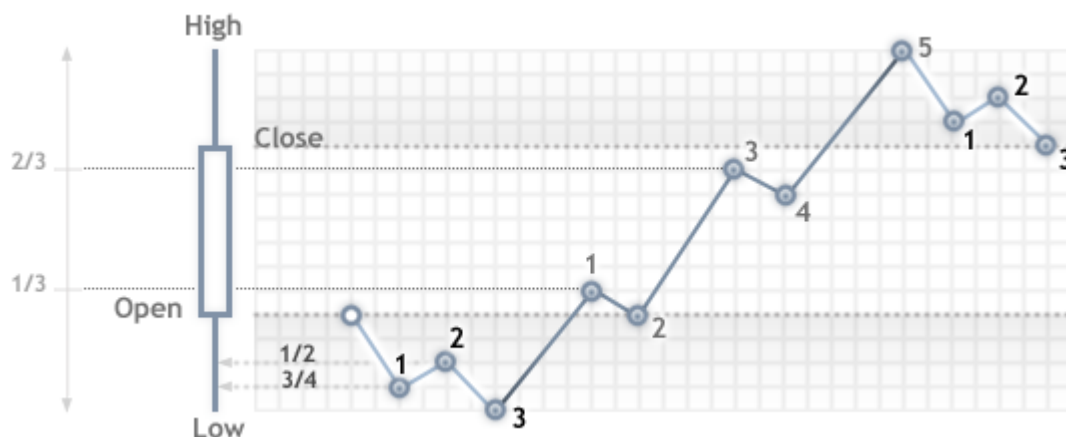


For this bar, we only know that the minute bar was opened on January 10th 2011 at 00 hours 00 minutes, but we know nothing about the seconds. It could have been opened at 00:00:12 or 00:00:36 (12 or 36 seconds after the start of a new day) or any other time within that minute. But we do know that the Open price of EURUSD was at 1.28940 at the opening time of the new minute bar.

We also don't know (accurate within a second) when we received the tick corresponding to the closing price of the considered minute bar. We know only one thing - the last Close price of the minute bar. For this minute, the price was 1.28958. The time of the appearance of High and Low prices is also unknown, but we know that the maximum and minimum prices were on the levels of 1.28958 and 1.28940, respectively.

To test the trading strategy, we need a sequence of ticks, on which the work of the Expert Advisor will be simulated. Thus, for every minute bar, we know the **4 control points**, where the price has definitely been. If a bar has only 4 ticks, then this is enough information to perform a testing, but usually the tick volume is greater than 4.

Hence, there is a need to generate additional control points for ticks, which occurred between the Open, High, Low, and Close prices. The principle of the "Every tick" ticks generation mode is described in the [The Algorithm of Ticks' Generation within the Strategy Tester of the MetaTrader 5 Terminal](#) a figure from which is presented below.



When testing in the "Every tick" mode, the [OnTick\(\)](#) function of the EA will be called at every control point. Each control point is a tick from a generated sequence. The EA will receive the time and price of the simulated tick, just as it would when working online.

Important: the "Every tick" testing mode is the most accurate, but at the same time, the most time consuming. For an initial testing of the majority of trading strategies, it is usually sufficient to use one of the other two testing modes.

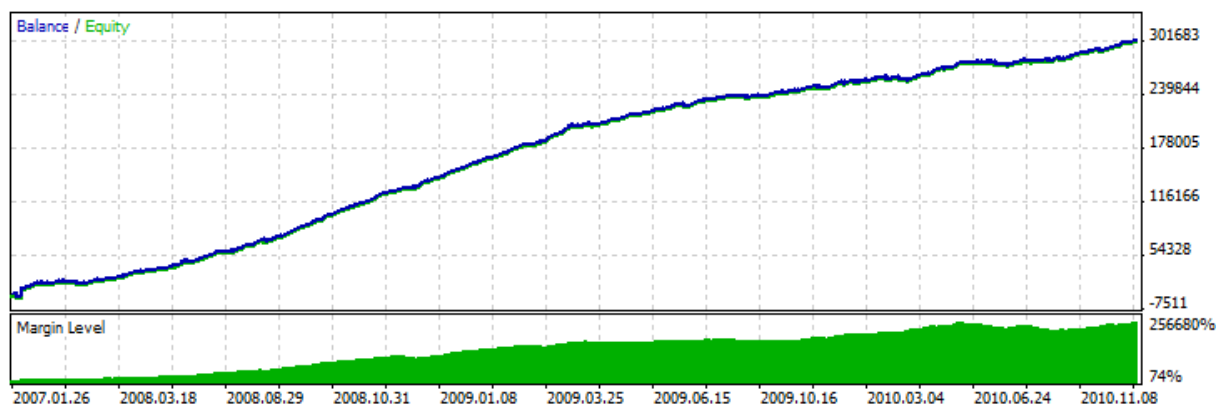
"1 Minute OHLC"

The "Every tick" mode is the most accurate of the three modes, but at the same time, is the slowest. The running of the [OnTick\(\)](#) handler occurs at every tick, while tick volume can be quite large. For a strategy, in which the tick sequence of price movement throughout the bar, does not matter, there is a faster and rougher simulation mode - "1 minute OHLC".

In the "1 minute OHLC" mode, the tick sequence is constructed only by the **OHLC prices of the minute bars**, the number of the generated control points is significantly reduced - hence, so is the testing time. The launch of the [OnTick\(\)](#) function is performed on all control points, which are constructed by the prices of OHLC minute bars.

The refusal to generate additional intermediate ticks between the Open, High, Low, and Close prices, leads to an appearance of rigid determinism in the development of prices, from the moment that the Open price is determined. This makes it possible to create a "Testing Grail", which shows a nice upward graph of the testing balance.

An example of such Grail is presented in the CodeBase - [Grr-al](#).



The figure shows a very attractive graph of this EA testing. How was it obtained? We know 4 prices for a minute bar, and we also know that the first is the Open price, and the last is the Close price. We have the High and Low prices between them, and the sequence of their occurrence is unknown, but it is known, that the High price is greater than or equal to the Open price (and the Low price is less than or equal to the Open price).

It is sufficient enough to determine the moment of receiving the Open price, and then analyze the next tick in order to determine what price we have at the moment - either the High or the Low. If the price is below the Open price, then we have a Low price and buy at this tick, the next tick will correspond to the High price, at which we will close the buy and open for sell. The next tick is the last one, this is the Close price, and we close the sale on it.

If after the price, we receive a tick with a price greater than the opening price, then the sequence of deals is reversed. Process a minute bar in this "cheat" mode, and wait for the next one.

When testing such EA on the history, everything goes smoothly, but once we launch it online, the truth begins to get revealed - the balance line remains steady, but heads downwards. To expose this trick, we simply need to run the EA in the "Every tick" mode.

Note: If the test results of the EA in the rough testing modes ("1 minute OHLC" and "Open Prices only") seem too good, make sure to test it in the "Every tick" mode.

"Open Prices Only"

In this mode ticks are generated based on the OHLC prices of the timeframe selected for testing. The OnTick() function of the Expert Advisor runs only at the beginning of the bar at the Open price. Due to this feature, stop levels and pending may trigger at a price that differs from the specified one (especially when testing on higher timeframes). Instead, we have an opportunity to quickly run an evaluation test of the Expert Advisor.

W1 and MN1 periods are the exceptions in the "Open Price Only" ticks generation mode: for these timeframes ticks are generated for the OHLC prices of each day, not OHLC prices of the week or month.

Suppose we test an Expert Advisor on EURUSD H1 in the "Open Prices Only" mode. In this case the total number of ticks (control points) will be no more than 4*number of one-hour bars within the tested interval. But the OnTick() handler is called only at the opening of the one-hour bar. The checks required for a correct testing occur on the rest of the ticks (that are "hidden" from the EA).

- The calculation of margin requirements;

- The triggering of Stop Loss and Take Profit levels;
- The triggering of pending orders;
- The removal of expired pending orders.

If there are no open positions or pending orders, we don't need to perform these checks on hidden ticks, and the increase of speed may be quite substantial. This "Open prices only" mode is well suited for testing strategies, which process deals only at the opening of the bar and do not use pending orders, as well as StopLoss and TakeProfit orders. For the class of such strategies, the necessary accuracy of testing is preserved.

Let's use the Moving Average Expert Advisor from the standard package as an example of an EA, which can be tested in any mode. The logic of this EA is built in such a way that all of the decisions are made at the opening of the bar, and deals are carried out immediately, without the use of pending orders.

Run a testing of the EA on EURUSD H1 on an interval from 2010.09.01 to 2010.12.31, and compare the graphs. The figure shows the balance graph from the test report for all of the three modes.



As you can see, the graphs on different testing modes are exactly the same for the Moving Average EA from the standard package.

There are some limitations on the "Open Prices Only" mode:

- You cannot use [the Random Delay execution mode](#).
- In the tested Expert Advisor, you cannot access data of the [timeframe](#) lower than that used for testing/optimization. For example, if you run testing/optimization on the H1 period, you can access data of H2, H3, H4 etc., but not M30, M20, M10 etc. In addition, the higher timeframes that are accessed must be multiple of the testing timeframe. For example, if you run testing in M20, you cannot access data of M30, but it is possible to access H1. These limitations are connected with the impossibility to obtain data of lower or non-multiple timeframes out of the bars generated during testing/optimization.

- Limitations on accessing data of other timeframes also apply to other symbols whose data are used by the Expert Advisor. In this case the limitation for each symbol depends on the first timeframe accessed during testing/optimization. Suppose, during testing on EURUSD H1, an Expert Advisor accesses data of GBPUSD M20. In this case the Expert Advisor will be able to further use data of EURUSD H1, H2, etc., as well as GBPUSD M20, H1, H2 etc.

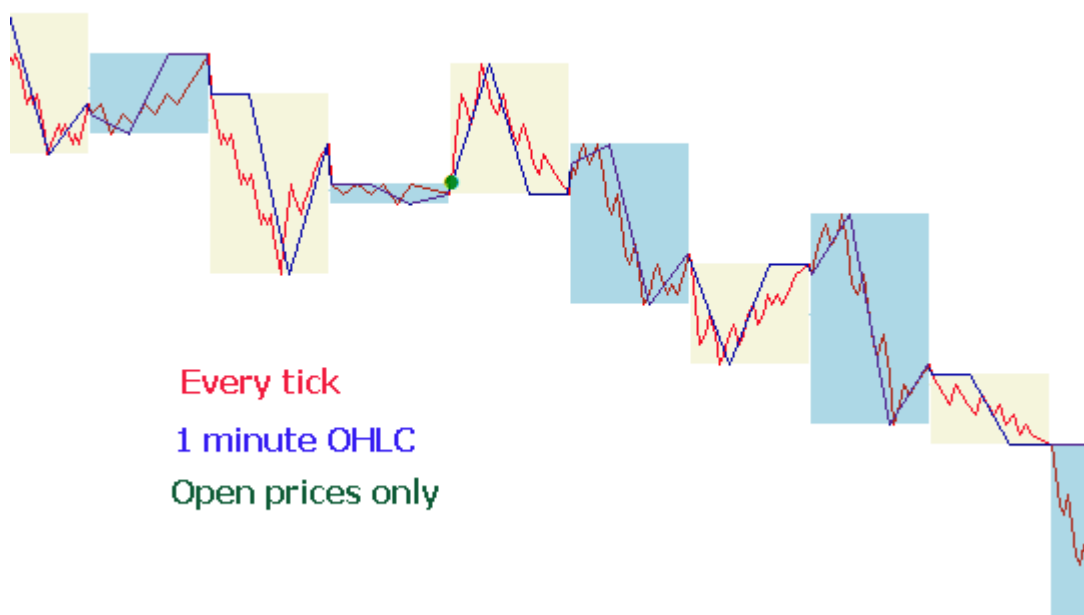
Note: The "Open prices only" mode has the fastest testing time, but it is not suitable for all of the trading strategies. Select the desired test mode based on the characteristics of the trading system.

To conclude the section on the tick generation modes, let's consider a visual comparison of the different tick generation modes for EURUSD, for two M15 bars on an interval from 2011.01.11 21:00:00 - 2011.01.11 21:30:00.

The ticks were saved into different files using the WriteTicksFromTester.mq5 EA and the ending of these files names are specified in filenameEveryTick, filenameOHLC and filenameOpenPrice [input-parameters](#).

Strategy Tester	Variable	Value
	<input type="checkbox"/> start	2011.01.11 21:00:00
	<input type="checkbox"/> end	2011.01.11 21:30:00
	<input checked="" type="checkbox"/> filenameEveryTick	everytick.csv
	<input checked="" type="checkbox"/> filenameOHLC	ohlc.csv
	<input checked="" type="checkbox"/> filenameOpenPrice	openprice.csv
Settings Inputs Agents Journal		

To obtain three files with three tick sequences (for each of the following modes "Every tick", "1 minute OHLC" and "Open prices only"), the EA was launched three times in the corresponding modes, in single runs. Then, the data from these three files were displayed on the chart using the TicksFromTester.mq5 indicator. The indicator code is attached to this article.



Every tick
1 minute OHLC
Open prices only

By default, all of the [file operations](#) in the MQL5 language are made within the "file sandbox", and during testing the EA has access only to its own "file sandbox". In order for the indicator and the EA to work with files from one folder during testing, we used the [flag FILE_COMMON](#). An example of code from the EA:

```
//--- open the file
file=FileOpen(filename,FILE_WRITE|FILE_CSV|FILE_COMMON,"");
//--- check file handle
if(file==INVALID_HANDLE)
{
    PrintFormat("Error in opening of file %s for writing. Error code=%d",filename,GetLastError());
    return;
}
else
{
    PrintFormat("The file will be created in %s folder",TerminalInfoString(TERMINAL_PATH));
}
```

For reading the data in the indicator, we also used the [flag FILE_COMMON](#). This allowed us to avoid manually transferring the necessary files from one folder to another.

```
//--- open the file
int file=FileOpen(fname,FILE_READ|FILE_CSV|FILE_COMMON,"");
//--- check file handle
if(file==INVALID_HANDLE)
{
    PrintFormat("Error in open of file %s for reading. Error code=%d",fname,GetLastError());
    return;
}
else
{
}
```



```
PrintFormat("File will be opened from %s",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
}
```

Simulation of spread

The price difference between the Bid and the Ask prices is called the spread. During testing, the spread is not modeled but is taken from historical data. If the spread is less than or equal to zero in the historical data, then the last known (at the moment of generation) spread is used by testing agent.

In the Strategy Tester, the spread is always considered floating. That is, [SymbolInfoInteger](#)(symbol, SYMBOL_SPREAD_FLOAT) always returns true.

In addition, the historical data contains tick values and trading volumes. For the storage and retrieval of data we use a special [MqlRates](#) structure:

```
struct MqlRates
{
    datetime time;           // Period start time
    double    open;          // Open price
    double    high;          // The highest price of the period
    double    low;           // The lowest price of the period
    double    close;         // Close price
    long      tick_volume;   // Tick volume
    int       spread;        // Spread
    long      real_volume;   // Trade volume
};
```

Using real ticks during a test

Testing and optimization on real ticks are as close to real conditions as possible. Instead of generated ticks based on minute data, it is possible to use real ticks accumulated by a broker. These are ticks from exchanges and liquidity providers.

To ensure the greatest test accuracy, minute bars are also used in the real ticks mode. The bars are applied to check and correct tick data. This also allows you to avoid the divergence of charts in the tester and the client terminal.

The tester compares the tick data to the minute bar parameters: a tick should not exceed the bar's High/Low levels, also initial and final ticks should coincide with the bar's Open/Close prices. The volume is compared as well. If a mismatch is detected, all ticks corresponding to this minute bar are discarded. Generated ticks are used instead (like in the "Every tick" mode).

If a symbol history has a minute bar with no tick data for it, the tester generates ticks in the "Every tick" mode. This allows plotting a correct chart in the tester in case a broker's tick data is insufficient.

If a symbol history has no minute bar but the appropriate tick data for the minute is present, the data can be used in the tester. For example, exchange symbol pairs are formed using Last prices. If only ticks with Bid/Ask prices without the Last price arrive from the server, the bar is not generated. The tester uses these tick data since they do not contradict the minute ones.

Tick data may not coincide with minute bars for various reasons, for example because of connection losses or other failures when transmitting data from a source to the client terminal. The minute data is considered more reliable during tests.

Keep in mind the following features when testing on real ticks:

- When launching a test, the minute data on a symbol is synchronized along with the tick one.
- Ticks are stored in the symbol cache of the strategy tester. The cache size does not exceed 128 000 ticks. When new ticks arrive, the oldest data is removed from the cache. However, the [CopyTicks](#) function allows receiving ticks outside the cache (only when testing on real ticks). In that case, the data is requested from the tester tick database that is completely similar to the corresponding client terminal database. No minute bar corrections are implemented to this base. Therefore, the ticks there may differ from the ones stored in the cache.

The Global Variables of the Client Terminal

During testing, the [global variables of the client terminal](#) are also emulated, but they are not related to the current [global variables of the terminal](#), which can be seen in the terminal using the F3 button. It means that all operations with the global variables of the terminal, during testing, take place outside of the client terminal (in the testing agent).

The Calculation of Indicators During Testing

In the real-time mode, the indicator values are calculated at every tick. The Strategy Tester adopted a cost-effective model for calculating indicators - [indicators are recalculated](#) only immediately before the running of the EA. It means that the recalculation of the indicators is done before the call of the `OnTick()`, `OnTrade()` and `OnTimer()` functions.

It does not matter whether or not there is a call for the indicator in a specific event handler. All the indicators with handles created by the [iCustom\(\)](#) or [IndicatorCreate\(\)](#) functions will be recalculated before calling the event handler.

Consequently, when testing in the "Every tick" mode, the calculation of the indicators takes place before the call of [OnTick\(\)](#) function.

If the timer is on in the EA, using the [EventSetTimer\(\)](#) function, then the indicators will be recalculated before each call of the [OnTimer\(\)](#) handler. Therefore, the testing time can be greatly increased with the use of an indicators, written in a non-optimal way.

Loading History during Testing

The history of a symbol to be tested is synchronized and loaded by the terminal from the trade server before starting the testing process. During the first time, the terminal loads all available history of a symbol in order not to request it later. Further only the new data are loaded.

A testing agent receives the history of a symbol to be tested from the client terminal right after the start of testing. If data of other instruments are used in the process of testing (for example, it is a multicurrency Expert Advisor), the testing agent requests the required history from the client terminal during the first call to such data. If historical data are available in the terminal, they are immediately

passed to the testing agent. If data are not available, the terminal requests and downloads them from the server, and then passes to the testing agent.

Data of additional instruments is also required for calculating cross-rates for trade operations. For example, when testing a strategy on EURCHF with the deposit currency in USD, prior to processing the first trading operation, the testing agent requests the history data of EURUSD and USDCHF from the client terminal, though the strategy does not contain direct use call of these symbols.

Before testing a multi-currency strategy, it is recommended to download all the necessary historical data to the client terminal. This will help to avoid delays in testing/optimization associated with download of the required data. You can download history, for example, by opening the appropriate charts and scrolling them to the history beginning. An example of forced loading of history into the terminal is available in the [Organizing Access to Data](#) section of the MQL5 Reference.

Testing agents, in turn, receive history from the terminal in the packed form. During the next testing, the tester does not load history from the terminal, because the required data is available since the previous run of the tester.

- The terminal loads history from a trade server only once, the first time the agent requests the history of a symbol to be tested from the terminal. The history is loaded in a packed form to reduce the traffic.
- Ticks are not sent over the network, they are generated on testing agents.

Multi-Currency Testing

The Strategy Tester allows us to perform a testing of strategies, trading on multiple symbols. Such EAs are conventionally referred to as multi-currency Expert Advisors, since originally, in the previous platforms, testing was performed only for a single symbol. In the Strategy Tester of the MetaTrader 5 terminal, we can model trading for all of the available symbols.

The tester loads the history of the used symbols from the **client terminal** (not from the trade server!) automatically during the first call of the symbol data.

The testing agent downloads only the missing history, with a small margin to provide the necessary data on the history, for the calculation of the indicators at the starting time of testing. For the time-frames D1 and less, the minimum volume of the downloaded history is one year.

Thus, if we run a testing on an interval 2010.11.01-2010.12.01 (testing for an interval of one month) with a period of M15 (each bar is equal to 15 minutes), then the terminal will be requested the history for the instrument for the entire year of 2010. For the weekly time-frame, we will request a history of 100 bars, which is about two years (a year has 52 weeks). For testing on a monthly time-frame the agent will request the history of 8 years (12 months x 8 years = 96 months).

If there isn't necessary bars, the **starting date of testing will be automatically shifted** from past to present to provide the necessary reserve of bars before the testing.

During testing, the "[Market Watch](#)" is emulated as well, from which one can obtain [information on symbols](#).

By default, at the beginning of testing, there is only one symbol in the "Market Watch" of the Strategy Tester - the symbol that the testing is running on. All of the necessary symbols are connected to the "Market Watch" of the Strategy Tester (not terminal!) automatically when referred to.

Prior to starting testing of a multi-currency Expert Advisor, it is necessary to select symbols required for testing in the "Market Watch" of the terminal and [load the required data](#). During the first call of a "foreign" symbol, its history will be automatically synchronized between the testing agent and the client terminal. A "foreign" symbol is the symbol other than that on which testing is running.

Referral to the data of an "other" symbol occurs in the following cases:

- When using the [technical indicators function](#) and [IndicatorCreate\(\)](#) on the symbol/timeframe;
- The request to the "Market Watch" data for the other symbol:
 1. [SeriesInfoInteger](#)
 2. [Bars](#)
 3. [SymbolSelect](#)
 4. [SymbolsSynchronized](#)
 5. [SymbolInfoDouble](#)
 6. [SymbolInfoInteger](#)
 7. [SymbolInfoString](#)
 8. [SymbolInfoTick](#)
 9. [SymbolInfoSessionQuote](#)
 10. [SymbolInfoSessionTrade](#)
 11. [MarketBookAdd](#)
 12. [MarketBookGet](#)
- Request of the time-series for a symbol/timeframe by using the following functions:
 1. [CopyBuffer](#)
 2. [CopyRates](#)
 3. [CopyTime](#)
 4. [CopyOpen](#)
 5. [CopyHigh](#)
 6. [CopyLow](#)
 7. [CopyClose](#)
 8. [CopyTickVolume](#)
 9. [CopyRealVolume](#)
 10. [CopySpread](#)

At the moment of the first call to an "other" symbol, the testing process is stopped and the history is downloaded for the symbol/timeframe, from the terminal to the testing agent. At the same time, the generation of tick sequence for this symbol is made.

An individual tick sequence is generated for each symbol, according to the selected tick generation mode. You can also request the history explicitly for the desired symbols by calling the [SymbolSelect\(\)](#) in the OnInit() handler - the downloading of the history will be made immediately prior to the testing of the Expert Advisor.

Thus, it does not require any extra effort to perform multi-currency testing in the MetaTrader 5 client terminal. Just open the charts of the appropriate symbols in the client terminal. The history will be automatically uploaded from the trading server for all the required symbols, provided that it contains this data.

Simulation of Time in the Strategy Tester

During testing, the local time [TimeLocal\(\)](#) is always equal to the server time [TimeTradeServer\(\)](#). In turn, the server time is always equal to the time corresponding to the GMT time - [TimeGMT\(\)](#). This way, all of these functions display the same time during testing.

The lack of a difference between the GMT, the Local, and the server time in the Strategy Tester is done deliberately in case there is no connection to the server. The test results should always be the same, regardless of whether or not there is a connection. Information about the server time is not stored locally, and is taken from the server.

Graphical Objects in Testing

During testing/optimization graphical objects are not plotted. Thus, when referring to the properties of a created object during testing/optimization, an Expert Advisor will receive zero values.

This limitation does not apply to testing in visual mode.

The OnTimer() Function in the Strategy Tester

MQL5 provides the opportunity for handling timer events. The call of the [OnTimer\(\)](#) handler is done regardless of the test mode. This means that if a test is running in the "Opening prices only" mode for the period H4, and the EA has a timer set to a call per second, then at the opening of each H4 bar, the OnTick() handler will be called one time, and the OnTimer() handler will be called 14400 times (3600 seconds * 4 hours). The amount by which the testing time of the EA will be increased depends on the logic of the EA.

To check the dependence of the testing time from the given frequency of the timer, we have created a simple EA without any trading operations.

```
//--- input parameters
input int      timer=1;           // timer value, sec
input bool     timer_switch_on=true; // timer on
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- run the timer if timer_switch_on==true
    if(timer_switch_on)
```

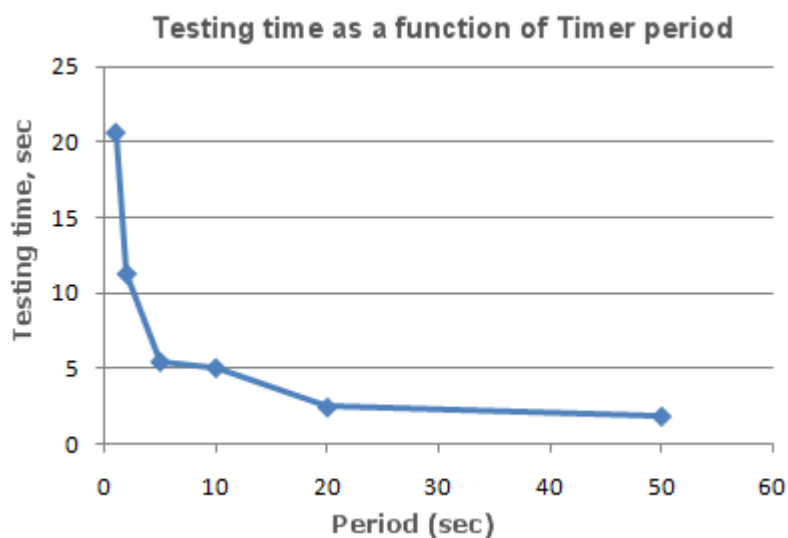


```

    {
        EventSetTimer(timer);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- stop the timer
    EventKillTimer();
}
//+-----+
//| Timer function |
//+-----+
void OnTimer()
{
//---
// take no actions, the body of the handler is empty
}
//+-----+

```

Testing time measurements were taken at different values of the timer parameter (periodicity of the Timer event). On the obtained data, we plot a testing time as function of Timer period.



It can be clearly seen that the smaller is the parameter timer, during the initialization of the [EventSetTimer](#)(Timer) function, the smaller is the period (Period) between the calls of the OnTimer() handler, and the larger is the testing time T, under the same other conditions.

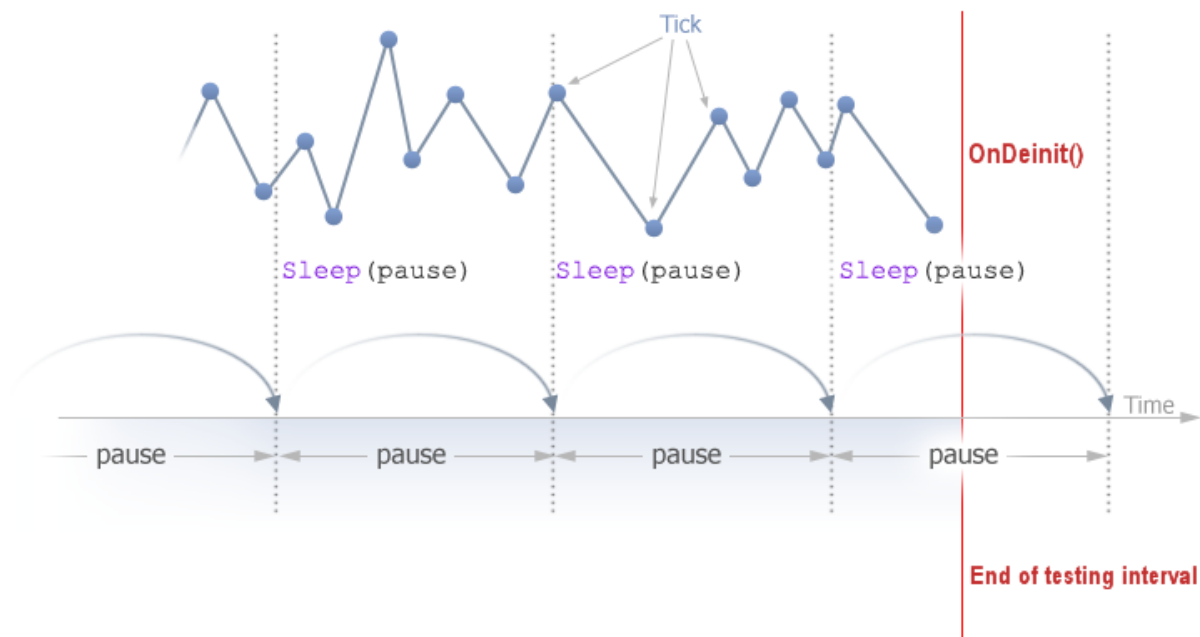
The Sleep() Function in the Strategy Tester

The [Sleep\(\)](#) function allows the EA or script to suspend the execution of the mql5-program for a while, when working on the graph. This can be useful when requesting data, which is not ready at the time of the request and you need to wait until it is ready. A detailed example of using the Sleep() function can be found in the section [Organizing Data Access](#).

The testing process is not lingered by the Sleep() calls. When you call the Sleep(), the generated ticks are "played" within a specified delay, which may result in the triggering of pending orders, stops, etc. After a Sleep() call, the simulated time in the Strategy Tester increases by an interval, specified in the parameter of the Sleep function.

If as a result of the execution of the Sleep() function, the current time in the Strategy Tester went over the testing period, then you will receive an error "Infinite Sleep loop detected while testing". If you receive this error, the test results are not rejected, all of the computations are performed in their full volume (the number of deals, subsidence, etc.), and the results of this testing are passed on to the terminal.

The Sleep() function will not work in OnDeinit(), since after it is called, the testing time will be guaranteed to surpass the range of the testing interval.



Using the Strategy Tester for Optimization Problems in Mathematical Calculations

The tester in the MetaTrader 5 terminal can be used, not only to testing trading strategies, but also for mathematical calculations. To use it, it's necessary to select the "Math calculations" mode:

In this case, only three functions will be called: `OnInit()`, `OnTester()`, `OnDeinit()`. In "Math calculations" mode the Strategy Tester doesn't generate any ticks and download the history.

The Strategy Tester works in "Math calculations" mode also if you specify the starting date greater than ending date.

When using the tester to solve mathematical problems, the uploading of the history and the generation of ticks does not occur.

A typical mathematical problem for solving in the MetaTrader 5 Strategy Tester - searching for an extremum of a function with many variables.

To solve it we need to:

- The calculation of function value should be located in `OnTester()` function;
- The function parameters must be defined as [input-variables](#) of the Expert Advisor;

Compile the EA, open the "Strategy Tester" window. In the "Input parameters" tab, select the required input variables, and define the set of parameter values by specifying the start, stop and step values for each of the function variables.

Select the optimization type - "Slow complete algorithm" (full search of parameters space) or "Fast genetic based algorithm". For a simple search of the extremum of the function, it is better to choose a fast optimization, but if you want to calculate the values for the entire set of variables, then it is best to use the slow optimization.

Select "Math calculation" mode and using the "Start" button, run the optimization procedure. Note that during the optimization the Strategy Tester will search for the maximum values of the `OnTester` function. To find a local minimum, return the inverse of the computed function value from the `OnTester` function:

```
return(1/function_value);
```

It is necessary to check that the `function_value` is not equal to zero, since otherwise we can obtain a [critical error](#) of dividing by zero.

There is another way, it is more convenient and does not distort the results of optimization, it was suggested by the readers of this article:

```
return(-function_value);
```

This option does not require the checking of the `function_value` for being equal to zero, and the surface of the optimization results in a 3D-representation has the same shape. The only difference is that it is mirrored comparing to the original.

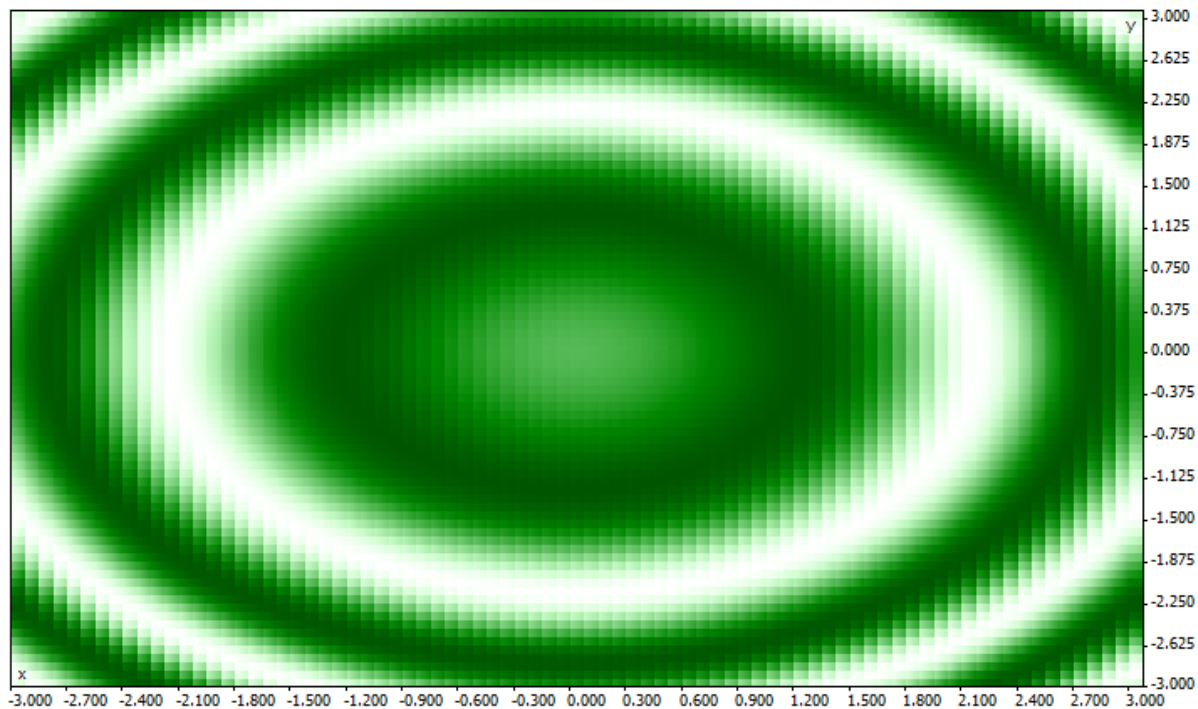
As an example, we provide the sink() function:

$$\text{sink}(x,y) = \sin(x^2 + y^2)$$

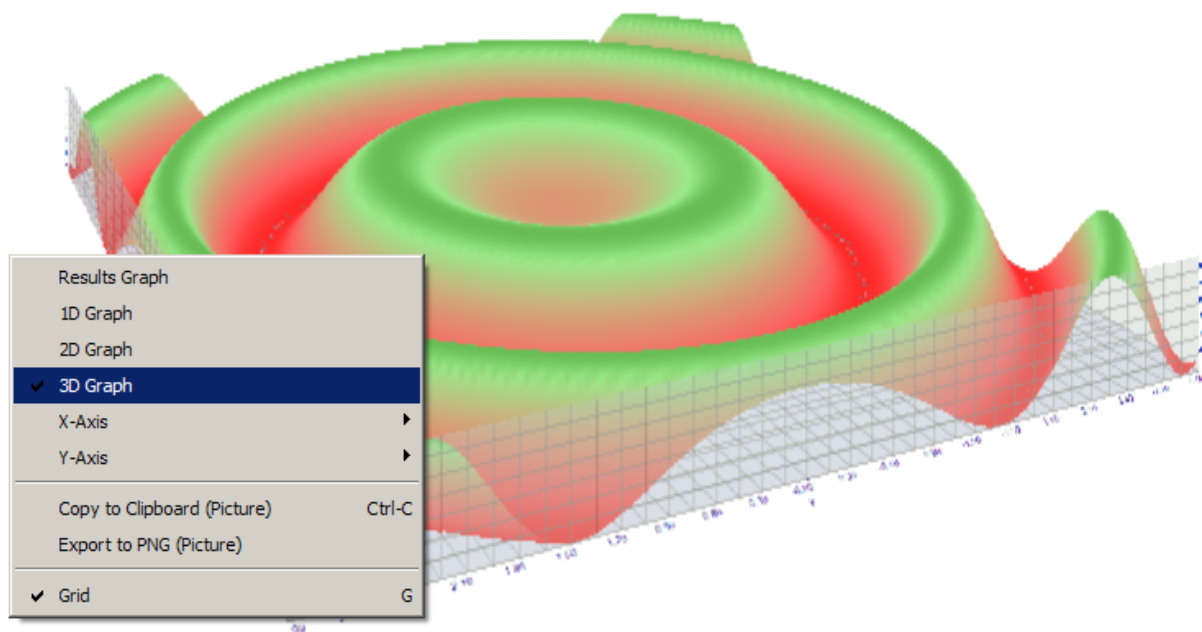
The code of the EA for finding the extremum of this function is placed into the OnTester():

```
//+-----+
//|                                     Sink.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input double   x=-3.0; // start=-3, step=0.05, stop=3
input double   y=-3.0; // start=-3, step=0.05, stop=3
//+-----+
//| Tester function                                     |
//+-----+
double OnTester()
{
//---
    double sink=MathSin(x*x+y*y);
//---
    return(sink);
}
//+-----+
```

Perform an optimization and see the [optimization results](#) in the form of a 2D graph.



The better the value is for a given pair of parameters (x, y), the more saturated the color is. As was expected from the view of the form of the `sink()` formula, its values forms concentric circles with a center at (0,0). One can see in the 3D-graph, that the `sink()` function has no single global extremum:



The Synchronization of Bars in the "Open prices only" mode

The tester in the MetaTrader 5 client terminal allows us to check the so-called "multi-currency" EAs. A multi-currency EA - is an EA that trades on two or more symbols.

The testing of strategies, that are trading on multiple symbols, imposes a few additional technical requirements on the tester:

- The generation of ticks for these symbols;
- The calculation of indicator values for these symbols;
- The calculation of margin requirements for these symbols;
- Synchronization of generated tick sequences for all trading symbols.

The Strategy Tester generates and plays a tick sequence for each instrument in accordance with the selected trading mode. At the same time, a [new bar](#) for each symbol is opened, regardless of how the bar opened on another symbol. This means that when testing a multi-currency EA, a situation may occur (and often does), when for one instrument a new bar has already opened, and for the other it has not. Thus, in testing, everything happens just like in reality.

This authentic simulation of the history in the tester does not cause any problems as long as the "Every tick" and "1 minute OHLC" testing modes are used. For these modes, enough ticks are generated for one candlestick, to be able to wait until the synchronization of bars from different symbols takes place. But how do we test multi-currency strategies in the "Open prices only" mode, if the synchronization of bars on trading instruments is mandatory? In this mode, the EA is called only on one tick, which corresponds to the time of the opening of the bars.

We'll illustrate it on an example: we are testing an EA on the EURUSD, and a new H1 candlestick has been opened on EURUSD. We can easily recognize this fact - while testing in the "Open prices only" mode, the [NewTick](#) event corresponds to the moment of a bar opening on the tested period. But there is no guarantee that the new candlestick was opened on the USDJPY symbol, which is used in the EA.

Under normal circumstances, it is sufficient enough to complete the work of the [OnTick\(\)](#) function and to check for the emergence of a new bar on USDJPY at the next tick. But when testing in the "Open prices only" mode, there will be no other tick, and so it may seem that this mode is not fit for testing multi-currency EAs. But this is not so - do not forget that the tester in MetaTrader 5 behaves just as it would in real life. You can wait until a new bar is opened on another symbols using the function [Sleep\(\)](#)!

The code of the EA `Synchronize_Bars_Use_Sleep.mq5`, which shows an example of the synchronization of bars in the "Open prices only" mode:

```
//+-----+
//|                                     Synchronize_Bars_Use_Sleep.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string  other_symbol="USDJPY";
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- check symbol
if(_Symbol==other_symbol)
{
```



```

        PrintFormat("You have to specify the other symbol in input parameters or select
        //--- forced stop testing
        return(INIT_PARAMETERS_INCORRECT);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function                                     |
//+-----+
void OnTick()
{
//--- static variable, used for storage of last bar time
    static datetime last_bar_time=0;
//--- sync flag
    static bool synchronized=false;
//--- if static variable isn't initialized
    if(last_bar_time==0)
    {
        //--- it's first call, save bar time and exit
        last_bar_time=(datetime)SeriesInfoInteger(_Symbol,Period(),SERIES_LASTBAR_DATE);
        PrintFormat("The last_bar_time variable is initialized with value %s",TimeToString(last_bar_time));
    }
//--- get open time of the last bar of chart symbol
    datetime curr_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DATE);
//--- if times aren't equal
    if(curr_time!=last_bar_time)
    {
        //--- save open bar time to the static variable
        last_bar_time=curr_time;
        //--- not synchronized
        synchronized=false;
        //--- print message
        PrintFormat("A new bar has appeared on symbol %s at %s",_Symbol,TimeToString(curr_time));
    }
//--- open time of the other symbol's bar
    datetime other_time;
//--- loop until the open time of other symbol become equal to curr_time
    while(!(curr_time==(other_time=(datetime)SeriesInfoInteger(other_symbol,Period(),SERIES_LASTBAR_DATE))))
    {
        PrintFormat("Waiting 5 seconds..");
        //--- wait 5 seconds and call SeriesInfoInteger(other_symbol,Period(),SERIES_LASTBAR_DATE)
        Sleep(5000);
    }
//--- bars are synchronized
    synchronized=true;
    PrintFormat("Open bar time of the chart symbol %s: is %s",_Symbol,TimeToString(last_bar_time));
    PrintFormat("Open bar time of the symbol %s: is %s",other_symbol,TimeToString(other_time));
//--- TimeCurrent() is not useful, use TimeTradeServer()

```



```
Print("The bars are synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS))
}
//+-----+
```

Notice the last line in the EA, which displays the current time when the fact of synchronization was established:

```
Print("The bars synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS));
```

To display the current time we used the [TimeTradeServer\(\)](#) function rather than [TimeCurrent\(\)](#). The TimeCurrent() function returns the time of the last tick, which does not change after using Sleep(). Run the EA in the "Open prices only" mode, and you will see a message about the synchronization of the bars.

Core 1	2010.12.01 20:00:05	The bars are synchronized at 2010.12.01 20:00:05
Core 1	2010.12.01 20:00:05	Open bar time of the chart symbol USDJPY: 2010.12.01 20:00
Core 1	2010.12.01 20:00:05	A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1	2010.12.01 20:00:00	Waiting 5 seconds..
Core 1	2010.12.01 20:00:05	A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1	2010.12.01 16:00:05	The bars are synchronized at 2010.12.01 16:00:05

Use the TimeTradeServer() function instead of the TimeCurrent(), if you need to obtain the current server time, and not the time of the last tick arrival.

There is another way to synchronize bars - using a timer. An example of such an EA is Synchronize_Bars_Use_OnTimer.mq5, which is attached to this article.

The IndicatorRelease() function in the Tester

After completing a single testing, a chart of the instrument is automatically opened, which displays the completed deals and the indicators used in the EA. This helps to visually check the entry and exit points, and compare them with the values of the indicators.

Note: indicators, displayed on the chart, which automatically opens after the completion of the testing, are calculated anew after the completion of testing. Even if these indicators were used in the tested EA.

But in some cases, the programmer may want to hide the information on which indicators were involved in the trading algorithms. For example, the code of the EA is rented or sold as an executable file, without the provision of the source code. For this purpose, the IndicatorRelease() function is suitable.

If the terminal sets a template with the name tester.tpl in the directory/profiles/templates of the client terminal, then it will be applied to the opened chart. In its absence, the default template is applied. (default.tpl).

The [IndicatorRelease\(\)](#) function is originally intended for releasing the calculating portion of the indicator, if it is no longer needed. This allows you to save both the memory and the CPU resources, because each tick calls for an indicator calculation. Its second purpose is to prohibit the showing of an indicator on the testing chart, after a single test run.

To prohibit the showing of the indicator on the chart after testing, call the [IndicatorRelease\(\)](#) with the handle of the indicator in the handler [OnDeinit\(\)](#). The OnDeinit() function is always called after the completion and before the showing of the testing chart.

```
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
    bool hidden=IndicatorRelease(handle_ind);
    if(hidden) Print("IndicatorRelease() successfully completed");
    else Print("IndicatorRelease() returned false. Error code ",GetLastError());
}
```

In order to prohibit the showing of the indicator on the chart, after the completion of a single test, use the function IndicatorRelease() in the handler OnDeinit().

Event Handling in the Tester

The presence of the OnTick() handler in the EA is not mandatory in order for it to be subjected to testing on historical data in the MetaTrader 5 tester. It is sufficient enough for the EA to contain at least one of the following function-handlers:

- [OnTick\(\)](#) - Event handler of a new tick arrival;
- [OnTrade\(\)](#) - Trading event handler;
- [OnTimer\(\)](#) - Event handler of a signal arrival from the timer;
- [OnChartEvent\(\)](#) - a handler for client events.

When testing in an EA, we can handle custom events using the [OnChartEvent\(\)](#) function, but in the indicators, this function can not be called in the tester. Even if the indicator has the [OnChartEvent\(\)](#) event handler and this indicator is used in the tested EA, the indicator itself will not receive any custom events.

During testing, an Indicator can generate custom events using the [EventChartCustom\(\)](#) function, and the EA can process this event in the OnChartEvent().

In addition to these events, special events associated with the process of testing and optimization are generated in the strategy tester:

- Tester - this event is generated after completion of Expert Advisor testing on history data. The Tester event is handled using the [OnTester\(\)](#) function. This function can be used only when testing Expert Advisor and is intended primarily for the calculation of a value that is used as a Custom max criterion for genetic optimization of input parameters.
- TesterInit - this event is generated during the start of optimization in the strategy tester before the very first pass. The TesterInit event is handled using the [OnTesterInit\(\)](#) function. During the start of optimization, an Expert Advisor with this handler is automatically loaded on a separate terminal chart with the symbol and period specified in the tester, and receives the TesterInit event. The function is used to initiate an Expert Advisor before start of optimization for further [processing of optimization results](#).

- **TesterPass** - this event is generated when a new [data frame](#) is received. The TesterPass event is handled using the [OnTesterPass\(\)](#) function. An Expert Advisor with this handler is automatically loaded on a separate terminal chart with the symbol/period specified for testing, and receives the TesterPass event when a frame is received during optimization. The function is used for dynamic handling of [optimization results](#) "on the spot" without waiting for its completion. Frames are added using the [FrameAdd\(\)](#) function, which can be called after the end of a single pass in the [OnTester\(\)](#) handler.
- **TesterDeinit** - this event is generated after the end of Expert Advisor optimization in the strategy tester. The TesterDeinit event is handled using the [OnTesterDeinit\(\)](#) function. An Expert Advisor with this handler is automatically loaded on a chart at the start of optimization, and receives TesterDeinit after its completion. The function is used for final processing of all [optimization results](#).

Testing Agents

Testing in the MetaTrader 5 client terminal is carried out using [test agents](#). Local agents are created and enabled automatically. The default number of local agents corresponds to the number of cores in a computer.

Each testing agent has its own copy of the [global variables](#), which is not related to the client terminal. The terminal itself is the dispatcher, which distributes the tasks to the local and remote agents. After executing a task on the testing of an EA, with the given parameters, the agent returns the results to the terminal. With a single test, only one agent is used.

The agent stores the history, received from the terminal, in separate folders, by the name of the instrument, so the history for EURUSD is stored in a folder named EURUSD. In addition, the history of the instruments is separated by their sources. The structure for storing the history looks the following way:

```
tester_catalog\Agent-IPaddress-Port\bases\name_source\history\symbol_name
```

For example, the history for EURUSD from the server MetaQuotes-Demo can be stored in the folder `tester_catalog\Agent-127.0.0.1-3000\bases\MetaQuotes-Demo\EURUSD`.

A local agent, after the completion of testing, goes into a standby mode, awaiting for the next task for another 5 minutes, so as not to waste time on launching for the next call. Only after the waiting period is over, the local agent shuts down and unloads from the CPU memory.

In case of an early completion of the testing, from the user's side (the "Cancel" button), as well as with the closing of the client terminal, all local agents immediately stop their work and are unloaded from the memory.

The Data Exchange between the Terminal and the Agent

When you run a test, the client terminal prepares to send an agent a number of parameter blocks:

- Input parameters for testing (simulation mode, the interval of testing, instruments, optimization criterion, etc.)
- The list of the selected symbols in the "Market Watch"
- The specification of the testing symbol (the contract size, the allowable margins from the market for setting a StopLoss and Takeprofit, etc)

- The Expert Advisor to be tested and the values of its input parameters
- Information about additional files (libraries, indicators, data files - [# property tester_...](#))

tester_indicator	string	Name of a custom indicator in the <code>format</code> of "indicator_name.ex5". Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a constant string. For all other cases (use of the IndicatorCreate() function or use of a non-constant string in the parameter that sets the indicator name) this property is required
tester_file	string	File name for a tester with the indication of extension, in double quotes (as a constant string). The specified file will be passed to tester. Input files to be tested, if there are necessary ones, must always be specified.
tester_library	string	Library name with the extension, in double quotes. A library can have extension dll or ex5. Libraries that require testing are defined automatically. However, if any of libraries is used by a custom indicator, this property is required

For each block of parameters, a digital fingerprint in the form of MD5-hash is created, which is sent to the agent. MD5-hash is unique for each set, its volume is many more times smaller than the amount of information on which it is calculated.

The agent receives a hash of blocks and compares them with those that it already has. If the fingerprint of the given parameter block is not present in the agent, or the received hash is different from the existing one, the agent requests this block of parameters. This reduces the traffic between the terminal and the agent.

After the testing, the agent returns to the terminal all of the results of the run, which are shown in the tabs "Test Results" and "Optimization Results": the received profit, the number of deals, the Sharpe coefficient, the result of the `OnTester()` function, etc.

During optimizing, the terminal hands out testing tasks to the agents in small packages, each package contains several tasks (each task means single testing with a set of input parameters). This reduces the exchange time between the terminal and the agent.

The agents never record to the hard disk the EX5-files, obtained from the terminal (EA, indicators, libraries, etc.) for security reasons, so that a computer with a running agent could not use the sent data. All other files, including DLL, are recorded in the sandbox. In remote agents you can not test EAs using DLL.

The testing results are added up by the terminal into a special cache of results (the result cache), for a quick access to them when they are needed. For each set of parameters, the terminal searches the result cache for already available results from the previous runs, in order to avoid re-runs. If the result with such a set of parameters is not found, the agent is given the task to conduct the testing.

All traffic between the terminal and the agent is encrypted.

Ticks are not sent over the network, they are generated on testing agents.

Using the Shared Folder of All of the Client Terminals

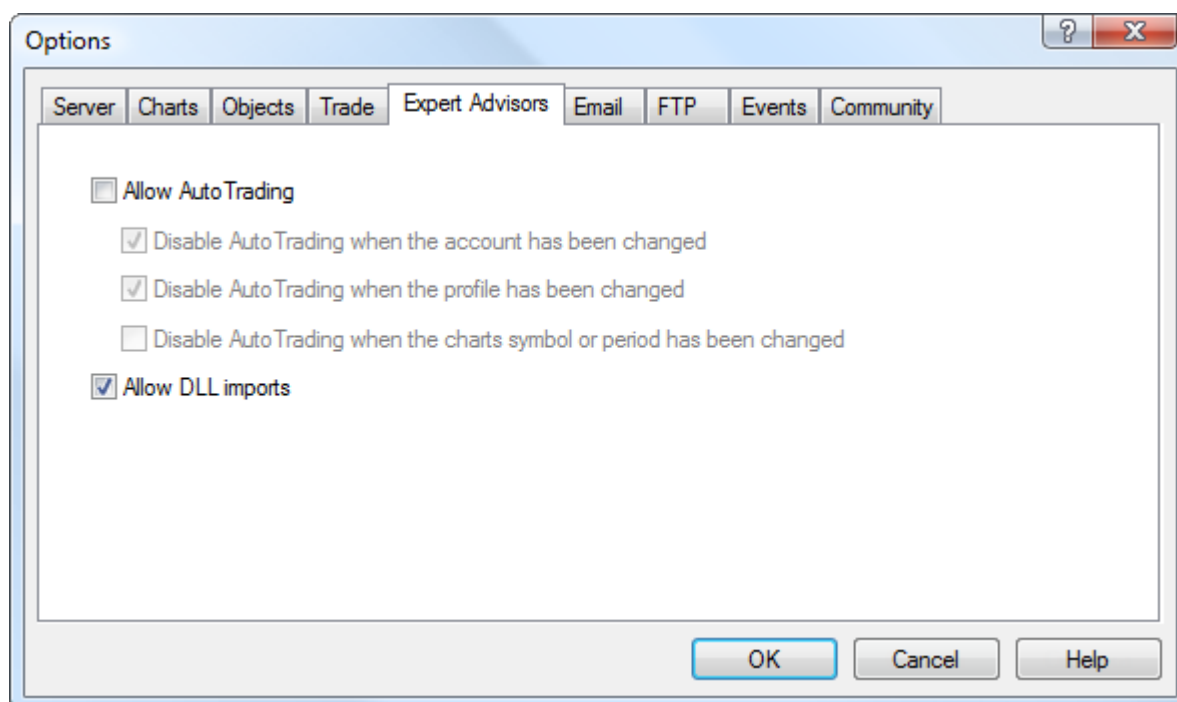
All testing agents are isolated from each other and from the client terminal: each agent has its own folder in which its logs are recorded. In addition, all file operations during the testing of the agent occur in the folder **agent_name/MQL5/Files**. However, we can implement the interaction between the local agents and the client terminal through a shared folder for all of the client terminals, if during the file opening you specify the flag [FILE_COMMON](#):

```
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //-- the shared folder for all of the client terminals
    common_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
    //-- draw out the name of this folder
    PrintFormat("Open the file in the shared folder of the client terminals %s", common_folder);
    //-- open a file in the shared folder (indicated by FILE_COMMON flag)
    handle=FileOpen(filename,FILE_WRITE|FILE_READ|FILE_COMMON);
    ... further actions
    //--
    return(INIT_SUCCEEDED);
}
```

Using DLLs

To speed up the optimization we can use not only local, but also [remote agents](#). In this case, there are some limitations for remote agents. First of all, remote agents do not display in their logs the results of the execution of the [Print\(\)](#) function, messages about the opening and closing of positions. A minimum of information is displayed in the log to prevent incorrectly written EAs from trashing up the computer, on which the remote agent is working, with messages.

A second limitation - the prohibition on the use of DLL when testing EAs. DLL calls are absolutely forbidden on remote agents for security reasons. On local agent, DLL calls in tested EAs are allowed only with the appropriate permission "Allow import DLL".



Note: When using 3rd party EAs (scripts, indicators) that require allowed DLL calls, you should be aware of the risks which you assume when allowing this option in the settings of the terminal. Regardless of how the EA will be used - for testing or for running on a chart.

The predefined Variables

For each executable mql5-program a set of predefined variables is supported, which reflect the state of the current price chart by the moment a mql5-program (Expert Advisor, script or custom indicator) is started.

Values of predefined variables are set by the client terminal before a mql5-program is started. Predefined variables are constant and cannot be changed from a mql5-program. As exception, there is a special variable `_LastError`, which can be reset to 0 by the [ResetLastError](#) function.

Variable	Value
_Digits	Number of decimal places
_Point	Size of the current symbol point in the quote currency
_LastError	The last error code
_Period	Timeframe of the current chart
_RandomSeed	Current status of the generator of pseudo-random integers
_StopFlag	Program stop flag
_Symbol	Symbol name of the current chart
_UninitReason	Uninitialization reason code

Predefined variables cannot be defined in a library. A library uses such variables that are defined in program from which this library is called.

int _Digits

The _Digits variable stores number of digits after a decimal point, which defines the price accuracy of the symbol of the current chart.

You may also use the [Digits\(\)](#) function.

double _Point

The _Point variable contains the point size of the current symbol in the quote currency.

You may also use the [Point\(\)](#) function.

int _LastError

The _LastError variable contains code of the last [error](#), that occurred during the mql5-program run. Its value can be reset to zero by [ResetLastError\(\)](#).

To obtain the code of the last error, you may also use the [GetLastError\(\)](#) function.

int _Period

The _Period variable contains the value of the timeframe of the current chart.

Also you may use the [Period\(\)](#) function.

See also

[PeriodSeconds](#), [Chart timeframes](#), [Date and Time](#), [Visibility of objects](#)

RandomSeed

Variable for storing the current state when generating pseudo-random integers. [_RandomSeed](#) changes its value when calling [MathRand\(\)](#). Use [MathSrand\(\)](#) to set the required initial condition.

x random number received by MathRand() function is calculated in the following way at each call:

```
x=_RandomSeed*214013+2531011;  
_RandomSeed=x;  
x=(x>>16)&0x7FFF;
```

See also

[MathRand\(\)](#), [MathSrand\(\)](#), [Integer types](#)

bool _StopFlag

The _StopFlag variable contains the flag of the mql5-program stop. When the client terminal is trying to stop the program, it sets the _StopFlag variable to true.

To check the state of the _StopFlag you may also use the [IsStopped\(\)](#) function.

string _Symbol

The _Symbol variable contains the symbol name of the current chart.

You may also use the [Symbol\(\)](#) function.

int _UninitReason

The _UninitReason variable contains the code of the program [uninitialization reason](#).

Usually, this code is obtained by [UninitializeReason\(\)](#) the function.

Common Functions

General-purpose functions not included into any specialized group are listed here.

Function	Action
Alert	Displays a message in a separate window
CheckPointer	Returns the type of the object pointer
Comment	Outputs a comment in the left top corner of the chart
CryptEncode	Transforms the data from array with the specified method
CryptDecode	Performs the inverse transformation of the data from array
DebugBreak	Program breakpoint in debugging
ExpertRemove	Stops Expert Advisor and unloads it from the chart
GetPointer	Returns the object pointer
GetTickCount	Returns the number of milliseconds that have elapsed since the system was started
GetMicrosecondCount	Returns the number of microseconds that have elapsed since the start of MQL5 program
MessageBox	Creates, displays a message box and manages it
PeriodSeconds	Returns the number of seconds in the period
PlaySound	Plays a sound file
Print	Displays a message in the log
PrintFormat	Formats and prints the sets of symbols and values in a log file in accordance with a preset format
ResetLastError	Sets the value of a predetermined variable _LastError to zero
ResourceCreate	Creates an image resource based on a data set
ResourceFree	Deletes dynamically created resource (freeing the memory allocated for it)
ResourceReadImage	Reads data from the graphical resource created by ResourceCreate() function or saved in EX5 file during compilation
ResourceSave	Saves a resource into the specified file

<u>SetUserError</u>	Sets the predefined variable _LastError into the value equal to ERR_USER_ERROR_FIRST + user_error
<u>SendFTP</u>	Sends a file at the address specified in the settings window of the "FTP" tab
<u>SendMail</u>	Sends an email at the address specified in the settings window of the "Email" tab
<u>SendNotification</u>	Sends push notifications to mobile terminals, whose MetaQuotes ID are specified in the "Notifications" tab
<u>Sleep</u>	Suspends execution of the current Expert Advisor or script within a specified interval
<u>TerminalClose</u>	Commands the terminal to complete operation
<u>TesterStatistics</u>	It returns the value of a specified statistic calculated based on testing results
<u>TesterWithdrawal</u>	Emulates the operation of money withdrawal in the process of testing
<u>TranslateKey</u>	Returns a Unicode character by a virtual key code
<u>WebRequest</u>	Sends HTTP request to the specified server
<u>ZeroMemory</u>	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.

Alert

Displays a message in a separate window.

```
void Alert(  
    argument,      // first value  
    ...           // other values  
);
```

Parameters

argument

[in] Any values separated by commas. To split the information output in several lines you can use the line feed character "\n" or "\r\n". The number of parameters can not exceed 64.

Return Value

No return value.

Note

Arrays can't be passed to the Alert() function. Arrays should be output elementwise. Data of the double type are output with 8 digits after the decimal point, data of the float type are displayed with 5 digits after the decimal point. To output the real numbers with a different precision or in a scientific format, use the [DoubleToString\(\)](#) function.

Data of the bool type is output as "true" or "false" strings. Dates are output as YYYY.MM.DD HH:MM:SS. To display a date in another format use the [TimeToString\(\)](#) function. Data of the color type are output either as an R,G,B string or as a color name, if the color is present in a color set.

Alert() function does not work in the [Strategy Tester](#).

CheckPointer

The function returns the type of the object [pointer](#).

```
ENUM_POINTER_TYPE CheckPointer(
    object* anyobject    // object pointer
);
```

Parameters

anyobject
[in] Object pointer.

Return value

Returns a value from the [ENUM_POINTER_TYPE](#) enumeration.

Note

An attempt to call an incorrect pointer results in the [critical termination](#) of a program. That's why it's necessary to call the CheckPointer function before using a pointer. A pointer can be incorrect in the following cases:

- the pointer is equal to [NULL](#);
- the object has been deleted using the [delete](#) operator.

This function can be used for checking pointer validity. A non-zero value warranties that the pointer can be used for accessing.

Example:

```
//+-----+
//| Deletes list by deleting its elements |
//+-----+
void CMyList::Destroy()
{
    //--- service pointer for working in the loop
    CItem* item;
    //--- go through loop and try to delete dynamic pointers
    while(CheckPointer(m_items) != POINTER_INVALID)
    {
        item=m_items;
        m_items=m_items.Next();
        if(CheckPointer(item) == POINTER_DYNAMIC)
        {
            Print("Dynamic object ",item.Identifier()," to be deleted");
            delete (item);
        }
        else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
    }
    //---
}
```


See also

[Object Pointers](#), [Checking the Object Pointer](#), [Object Delete Operator delete](#)

Comment

This function outputs a comment defined by a user in the top left corner of a chart.

```
void Comment (
    argument,      // first value
    ...           // next values
);
```

Parameters

...

[in] Any values, separated by commas. To delimit output information into several lines, a line break symbol "\n" or "\r\n" is used. Number of parameters cannot exceed 64. Total length of the input comment (including invisible symbols) cannot exceed 2045 characters (excess symbols will be cut out during output).

Return Value

No return value

Note

Arrays can't be passed to the Comment() function. Arrays must be entered element-by-element.

Data of double type are output with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what notation will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the [DoubleToString\(\)](#) function.

Data of bool type are output as "true" or "false" strings. Dates are shown as YYYY.MM.DD HH:MI:SS. To show dates in another format, use the [TimeToString\(\)](#) function. Data of color type are output either as R,G,B string or as a color name, if this color is present in the color set.

Example:

```
void OnTick()
{
    //---
    double Ask,Bid;
    int Spread;
    Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
    Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
    //--- Output values in three lines
    Comment(StringFormat("Show prices\nAsk = %G\nBid = %G\nSpread = %d",Ask,Bid,Spread))
}
```

See also

[ChartSetString](#), [ChartGetString](#)

CryptEncode

Transforms the data from array with the specified method.

```
int CryptEncode(
    ENUM_CRYPT_METHOD  method,      // method
    const uchar&        data[],      // source array
    const uchar&        key[],       // key
    uchar&              result[]     // destination array
);
```

Parameters

method

[in] Data transformation method. Can be one of the values of [ENUM_CRYPT_METHOD](#) enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Return Value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the [error](#) call the [GetLastError\(\)](#) function.

Example:

```
//+-----+
//| ArrayToHex                                     |
//+-----+
string ArrayToHex(uchar &arr[],int count=-1)
{
    string res="";
//--- check
    if(count<0 || count>ArraySize(arr))
        count=ArraySize(arr);
//--- transform to HEX string
    for(int i=0; i<count; i++)
        res+=StringFormat("%.2X",arr[i]);
//---
    return(res);
}
//+-----+
//| Script program start function                 |
//+-----+
void OnStart()
```



```

{
    string text="The quick brown fox jumps over the lazy dog";
    string keystr="ABCDEFGH";
    uchar src[],dst[],key[];
    //--- prepare key
    StringToArray(keystr,key);
    //--- copy text to source array src[]
    StringToArray(text,src);
    //--- print initial data
    PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(src));
    //--- encrypt src[] with DES 56-bit key in key[]
    int res=CryptEncode(CRYPT_DES,src,key,dst);
    //--- check error
    if(res>0)
    {
        //--- print encrypted data
        PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
        //--- decode dst[] to src[]
        res=CryptDecode(CRYPT_DES,dst,key,src);
        //--- check error
        if(res>0)
        {
            //--- print decoded data
            PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToString(src));
        }
        else
            Print("Error in CryptDecode. Error code=",GetLastError());
    }
    else
        Print("Error in CryptEncode. Error code=",GetLastError());
}

```

See also

[Array Functions](#), [CryptDecode\(\)](#)

CryptDecode

Performs the inverse transformation of the data from array, tranformed by [CryptEncode\(\)](#).

```
int  CryptEncode(  
    ENUM_CRYPT_METHOD  method,          // method  
    const uchar&        data[],          // source array  
    const uchar&        key[],           // key  
    uchar&               result[]        // destination array  
);
```

Parameters

method

[in] Data transformation method. Can be one of the values of [ENUM_CRYPT_METHOD](#) enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Return Value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the [error](#) call the [GetLastError\(\)](#) function.

See also

[Array Functions](#), [CryptEncode\(\)](#)

DebugBreak

It is a program breakpoint in debugging.

```
void DebugBreak();
```

Return Value

No return value.

Note

Execution of an MQL5 program is interrupted only if a program is started in a debugging mode. The function can be used for viewing values of variables and/or for further step-by-step execution.

ExpertRemove

The function stops an [Expert Advisor](#) and unloads it from a chart.

```
void ExpertRemove();
```

Return Value

No return value.

Note

The Expert Advisor is not stopped immediately as you call `ExpertRemove()`; just a flag to stop the EA operation is set. That is, any next event won't be processed, [OnDeinit\(\)](#) will be called and the Expert Advisor will be unloaded and removed from the chart.

Example:

```
//+-----+
//|                                     Test_ExpertRemove.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20; // number of ticks before EA unload
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
    Print(TimeCurrent(), ": ", __FUNCTION__, " reason code = ", reason);
//--- "clear" comment
    Comment("");
//---
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
    static int tick_counter=0;
//---
    tick_counter++;
    Comment("\nBefore unloading expert advisor ", __FILE__, " left",
            (ticks_to_close-tick_counter), " ticks");
//--- before
    if(tick_counter>=ticks_to_close)
    {
```



```
ExpertRemove();  
Print(TimeCurrent(), ": ", __FUNCTION__, " expert advisor will be unloaded");  
}  
Print("tick_counter =", tick_counter);  
//---  
}  
//+-----+
```

See also

[Program running](#), [Client terminal events](#)

GetPointer

The function returns the object [pointer](#).

```
void* GetPointer(
    any_class anyobject    // object of any class
);
```

Parameters

anyobject

[in] Object of any class.

Return Value

The function returns the object pointer.

Note

Only class objects have pointers. Instances of [structures](#) and simple-type variables can't have pointers. The class object not created using the `new()` operator, but, e.g., automatically created in the array of objects, still has a pointer. But this pointer will be of the automatic type `POINTER_AUTOMATIC`, therefore the [delete\(\)](#) operator can't be applied to it. Aside from that, the type pointer doesn't differ from dynamic pointers of the [POINTER_AUTOMATIC](#) type.

Since variables of structure types and simple types do not have pointers, it's prohibited to apply the `GetPointer()` function to them. It's also prohibited to pass the pointer as a function argument. In all these cases the compiler will notify an error.

An attempt to call an incorrect pointer causes the [critical termination](#) of a program. That's why the [CheckPointer\(\)](#) function should be called prior to using a pointer. A pointer can be incorrect in the following cases:

- the pointer is equal to [NULL](#);
- the object has been deleted using the [delete](#) operator.

This function can be used to check the validity of a pointer. A non-zero value guarantees, that the pointer can be used for accessing.

Example:

```
//+-----+
//|                                     Check_GetPointer.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+-----+
//| Class implementing the list element |
//+-----+
```



```

class CItem
{
    int          m_id;
    string       m_comment;
    CItem*       m_next;
public:
    CItem() { m_id=0; m_comment=NULL; m_next=NULL; }
    ~CItem() { Print("Destructor of ",m_id,
                    (CheckPointer(GetPointer(this))==POINTER_DYNAMIC)
                    "dynamic":"non-dynamic"); }

    void          Initialize(int id,string comm) { m_id=id; m_comment=comm; }
    void          PrintMe() { Print(__FUNCTION__,":",m_id,m_comment); }
    int           Identifier() { return(m_id); }
    CItem*        Next() {return(m_next); }
    void          Next(CItem *item) { m_next=item; }
};

//+-----+
//| Simplest class of the list |
//+-----+

class CMyList
{
    CItem*        m_items;
public:
    CMyList() { m_items=NULL; }
    ~CMyList() { Destroy(); }

    bool          InsertToBegin(CItem* item);
    void          Destroy();
};

//+-----+
//| Inserts list element at the beginning |
//+-----+

bool CMyList::InsertToBegin(CItem* item)
{
    if(CheckPointer(item)==POINTER_INVALID) return(false);
//---
    item.Next(m_items);
    m_items=item;
//---
    return(true);
}

//+-----+
//| Deleting the list by deleting elements |
//+-----+

void CMyList::Destroy()
{
//--- service pointer to work in a loop
    CItem* item;
//--- go through the loop and try to delete dynamic pointers
    while(CheckPointer(m_items)!=POINTER_INVALID)

```



```

    {
        item=m_items;
        m_items=m_items.Next();
        if(CheckPointer(item)==POINTER_DYNAMIC)
        {
            Print("Dynamyc object ",item.Identifier()," to be deleted");
            delete (item);
        }
        else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
    }
}
//---
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    CMyList list;
    CItem items[10];
    CItem* item;
    //--- create and add into the list a dynamic object pointer
    item=new CItem;
    if(item!=NULL)
    {
        item.Initialize(100,"dynamic");
        item.PrintMe();
        list.InsertToBegin(item);
    }
    //--- add automatic pointers into the list
    for(int i=0; i<10; i++)
    {
        items[i].Initialize(i,"automatic");
        items[i].PrintMe();
        item=GetPointer(items[i]);
        if(CheckPointer(item)!=POINTER_INVALID)
            list.InsertToBegin(item);
    }
    //--- add one more dynamic object pointer at the list beginning
    item=new CItem;
    if(item!=NULL)
    {
        item.Initialize(200,"dynamic");
        item.PrintMe();
        list.InsertToBegin(item);
    }
    //--- delete all the list elements
    list.Destroy();
    //--- all the list elements will be deleted after the script is over
    //--- see the Experts tab in the terminal

```



```
}
```

See also

[Object Pointers](#), [Checking the Object Pointer](#), [Object Delete Operator delete](#)

GetTickCount

The `GetTickCount()` function returns the number of milliseconds that elapsed since the system start.

```
uint GetTickCount();
```

Return Value

Value of `uint` type.

Note

Counter is limited by the restrictions of the system timer. Time is stored as an unsigned integer, so it's overfilled every 49.7 days if a computer works uninterruptedly.

Example:

```
#define MAX_SIZE 40
//+-----+
//| Script for measuring computation time of 40 Fibonacci numbers |
//+-----+
void OnStart()
{
//--- Remember the initial value
    uint start=GetTickCount();
//--- A variable for getting the next number in the Fibonacci series
    long fib=0;
//--- In loop calculate the specified amount of numbers from Fibonacci series
    for(int i=0;i<MAX_SIZE;i++) fib=TestFibo(i);
//--- Get the spent time in milliseconds
    uint time=GetTickCount()-start;
//--- Output a message to the Experts journal
    PrintFormat("Calculating %d first Fibonacci numbers took %d ms",MAX_SIZE,time);
//--- Script completed
    return;
}
//+-----+
//| Function for getting Fibonacci number by its serial number |
//+-----+
long TestFibo(long n)
{
//--- The first member of the Fibonacci series
    if(n<2) return(1);
//--- All other members are calculated by the following formula
    return(TestFibo(n-2)+TestFibo(n-1));
}
```

See also

[Date and Time](#)

GetMicrosecondCount

The GetMicrosecondCount() function returns the number of microseconds that have elapsed since the start of MQL5-program.

```
ulong GetMicrosecondCount();
```

Return Value

Value of ulong type.

Example:

```
//+-----+
//| Test function |
//+-----+
void Test()
{
    int    res_int=0;
    double res_double=0;
//---
    for(int i=0;i<10000;i++)
    {
        res_int+=i*i;
        res_int++;
        res_double+=i*i;
        res_double++;
    }
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    uint    ui=0,ui_max=0,ui_min=INT_MAX;
    ulong   ul=0,ul_max=0,ul_min=INT_MAX;
//--- number of measurements
    for(int count=0;count<1000;count++)
    {
        uint ui_res=0;
        ulong ul_res=0;
//---
        for(int n=0;n<2;n++)
        {
            //--- select measurement type
            if(n==0) ui=GetTickCount();
            else     ul=GetMicrosecondCount();
            //--- execute code
            Test();
            //--- add measurement result (depending on type)
```



```
        if(n==0) ui_res+=GetTickCount()-ui;
        else      ul_res+=GetMicrosecondCount()-ul;
    }

    //--- calculate minimum and maximum time for both measurements
    if(ui_min>ui_res) ui_min=ui_res;
    if(ui_max<ui_res) ui_max=ui_res;
    if(ul_min>ul_res) ul_min=ul_res;
    if(ul_max<ul_res) ul_max=ul_res;
}

//---
Print("GetTickCount error(msec): ",ui_max-ui_min);
Print("GetMicrosecondCount error(msec): ",DoubleToString((ul_max-ul_min)/1000.0,2))
}
```

See also

[Date and Time](#)

MessageBox

It creates and shows a message box and manages it. A message box contains a message and header, any combination of predefined signs and command buttons.

```
int MessageBox(  
    string text,           // message text  
    string caption=NULL,  // box header  
    int flags=0           // defines set of buttons in the box  
);
```

Parameters

text

[in] Text, containing message to output.

caption=NULL

[in] Optional text to be displayed in the box header. If the parameter is empty, Expert Advisor name is shown in the box header.

flags=0

[in] Optional [flags](#) defining appearance and behavior of a message box. Flags can be a combination of a special group of flags.

Return Value

If the function is successfully performed, the returned value is one of values of [MessageBox\(\)](#) return codes.

Note

The function is not recommended for use in custom indicators, as the call to the MessageBox() suspends the indicator's [thread of execution](#) for the whole time while waiting for the user's response. And since all indicators for each symbol are executed in a single thread, all charts of all timeframes for this symbol will be suspended.

MessageBox() function does not work in the [Strategy Tester](#).

PeriodSeconds

This function returns number of seconds in a period.

```
int PeriodSeconds(  
    ENUM_TIMEFRAMES period=PERIOD_CURRENT    // chart period  
);
```

Parameters

period=PERIOD_CURRENT

[in] Value of a chart period from the enumeration [ENUM_TIMEFRAMES](#). If the parameter isn't specified, it returns the number of seconds of the current chart period, at which the program runs.

Return Value

Number of seconds in a selected period.

See also

[_Period](#), [Chart timeframes](#), [Date and Time](#), [Visibility of objects](#)

PlaySound

It plays a sound file.

```
bool PlaySound(  
    string filename    // file name  
);
```

Parameters

filename

[in] Path to a sound file. If filename=NULL, the playback is stopped.

Return Value

true - if the file is found, otherwise - false.

Note

The file must be located in terminal_directory\Sounds or its sub-directory. Only WAV files are played.

Call of PlaySound() with NULL parameter stops playback.

PlaySound() function does not work in the [Strategy Tester](#).

See also

[Resources](#)

Print

It enters a message in the Expert Advisor log. Parameters can be of any type.

```
void Print(
    argument,      // first value
    ...           // next values
);
```

Parameters

...

[in] Any values separated by commas. The number of parameters cannot exceed 64.

Note

Arrays cannot be passed to the Print() function. Arrays must be input element-by-element.

Data of double type are shown with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what entry will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the [PrintFormat\(\)](#) function.

Data of bool type are output as "true" or "false" lines. Dates are shown as YYYY.MM.DD HH:MI:SS. To show data in another format, use [TimeToString\(\)](#). Data of color type are returned either as R,G,B line or as a color name, if this color is present in the color set.

Print() function does not work during optimization in the [Strategy Tester](#).

Example:

```
void OnStart()
{
    //--- Output DBL_MAX using Print(), this is equivalent to PrintFormat("%.16G,DBL_MAX)
    Print("---- how DBL_MAX looks like ----");
    Print("Print(DBL_MAX)=",DBL_MAX);
    //--- Now output a DBL_MAX number using PrintFormat()
    PrintFormat("PrintFormat("%.16G,DBL_MAX)=%.16G",DBL_MAX);
    //--- Output to the Experts journal
    // Print(DBL_MAX)=1.797693134862316e+308
    // PrintFormat("%.16G,DBL_MAX)=1.797693134862316E+308

    //--- See how float is output
    float c=(float)M_PI; // We should explicitly cast to the target type
    Print("c=",c, "    Pi=",M_PI, "    (float)M_PI=",(float)M_PI);
    // c=3.14159    Pi=3.141592653589793    (float)M_PI=3.14159

    //--- Show what can happen with arithmetic operations with real types
    double a=7,b=200;
    Print("---- Before arithmetic operations");
    Print("a=",a, "    b=",b);
    Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
    //--- Divide a by b (7/200)
```



```

a=a/b;
//--- Now emulate restoring a value in the b variable
b=7.0/a; // It is expected that b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 - but it differs
//--- Output the newly calculated value of b
Print("----- After arithmetic operations");
Print("Print(b)=",b);
Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.999999999999716 (see that b is no more equal to 200)

//--- Create a very small value epsilon=1E-013
double epsilon=1e-13;
Print("----- Create a very small value");
Print("epsilon=",epsilon); // Get epsilon=1E-013
//--- Now subtract epsilon from b and again output the value to the Experts journal
b=b-epsilon;
//--- Use two ways
Print("----- After subtracting epsilon from the b variable");
Print("Print(b)=",b);
Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=199.9999999999999 (now the value of b after subtracting epsilon cannot be rounded to 200)
// Print(DoubleToString(b,16))=199.9999999999998578
// (now the value of b after subtracting epsilon cannot be rounded to 200)
}

```

See also

[DoubleToString](#), [StringFormat](#)

PrintFormat

It formats and enters sets of symbols and values in the Expert Advisor log in accordance with a preset format.

```
void PrintFormat(  
    string format_string,    // format string  
    ...                     // values of simple types  
);
```

Parameters

- format_string*
- [in] A format string consists of simple symbols, and if the format string is followed by arguments, it also contains format specifications.
- ...
- [in] Any values of simple types separated by commas. Total number of parameters can't exceed 64 including the format string.

Return Value

String.

Note

PrintFormat() function does not work during optimization in the [Strategy Tester](#).

The number, order and type of parameters must exactly match the set of qualifiers, otherwise the print result is undefined. Instead of PrintFormat() you can use [printf\(\)](#).

If the format string is followed by parameters, this string must contain format specifications that denote output format of these parameters. Specification of format always starts with the percent sign (%).

A format string is read from left to right. When the first format specification is met (if there is any), the value of the first parameter after the format string is transformed and output according to the preset specification. The second format specification calls transformation and output of the second parameter, and so on till the format string end.

The format specification has the following form:

%[flags][width][.precision][{h | l | ll | l32 | l64}]type

Each field of the format specification is either a simple symbol, or a number denoting a simple format option. The simplest format specification contains only the percent sign (%) and a symbol defining the [type of the output parameter](#) (for example, %s). If you need to output the percent sign in the format string, use the format specification %%.

flags

Flag	Description	Default Behavior
------	-------------	------------------

- (minus)	Left justification within the set width	Right justification
+ (plus)	Output of the + or - sign for values of sign types	The sign is shown only if the value is negative
0 (zero)	Zeroes are added before an output value within the preset width . If 0 flag is specified with an integer format (i, u, x, X, o, d) and accuracy specification is set (for example, %04.d), then 0 is ignored.	Nothing is added
space	A space is shown before an output value, if it is a sign and positive value	Spaces aren't inserted
#	If used together with the format o, x or X, then before the output value 0, 0x or 0X is added respectively.	Nothing is added
	If used together with the format e, E, a or A, value is always shown with a decimal point.	Decimal point is shown only if there is a non-zero fractional part.
	If used together with the format g or G, flag defines presence of a decimal point in the output value and prevents the cutting off of leading zeroes. Flag # is ignored when used together with formats c, d, i, u, s.	Decimal point is shown only if there is a non-zero fractional part. Leading zeroes are cut off.

width

A non-negative decimal number that sets the minimal number of output symbols of the formatted value. If the number of output symbols is less than the specified width, the corresponding number of spaces is added from the left or right depending on the alignment (flag -). If there is flag zero (0), the corresponding number of zeroes is added before the output value. If the number of output symbols is greater than the specified width, the output value is never cut off.

If an asterisk (*) is specified as width, value of int type must be indicated in the corresponding place of the list of passed parameters. It will be used for specifying width of the output value.

precision

A non-negative decimal number that sets the output precision - number of digits after a decimal point. As distinct from width specification, precision specification can cut off the part of fractional type with or without rounding.

The use of precision specification is different for different format [types](#).

Types	Description	Default Behavior
a, A	Precision specification sets the number of digits after a decimal point.	Default precision - 6.
c, C	Not used	
d, i, u, o, x, X	Sets minimal number of output digits. If number of digits in a corresponding parameter is less than this precision, zeroes are added to the left of the output value. The output value isn't cut off, if the number of output digits is larger than the specified precision.	Default precision - 1.
e, E, f	Sets number of output digits after a decimal point. The last digit is rounded off.	Default precision - 6. If set precision is 0 or decimal part is absent, the decimal point is not shown.
g, G	Sets maximal number of meaningful numbers.	6 meaningful numbers are output.
s, S	Sets number of output symbols of a string. If the string length exceeds the precision, the string is cut off.	The whole string is output.

h | l | ll | I32 | I64

Specification of data sizes, passed as a parameter.

Parameter Type	Used Prefix	Joint Specifier of Type
int	l (lower case L)	d, i, o, x, or X
uint	l (lower case L)	o, u, x, or X
long	ll (two lower case L)	d, i, o, x, or X
short	h	d, i, o, x, or X
ushort	h	o, u, x, or X

int	l32	d, i, o, x, or X
uint	l32	o, u, x, or X
long	l64	d, i, o, x, or X
ulong	l64	o, u, x, or X

type

Type specifier is the only obligatory field for formatted output.

Symbol	Type	Output Format
c	int	Symbol of short type (Unicode)
C	int	Symbol of char type (ANSI)
d	int	Signed decimal integer
i	int	Signed decimal integer
o	int	Unsigned octal integer
u	int	Unsigned decimal integer
x	int	Unsigned hexadecimal integer, using "abcdef"
X	int	Unsigned hexadecimal integer, using "ABCDEF"
e	double	A real value in the format [-] d.ddde[sign] ddd, where d - one decimal digit, dddd - one or more decimal digits, ddd - a three-digit number that determines the size of the exponent, sign - plus or minus
E	double	Similar to the format of e, except that the sign of exponent is output by upper case letter (E instead of e)
f	double	A real value in the format [-] dddd.dddd, where dddd - one or more decimal digits. Number of displayed digits before the decimal point depends on the size of number value. Number of digits after the decimal point

		depends on the required accuracy.
g	double	A real value output in f or e format depending on what output is more compact.
G	double	A real value output in F or E format depending on what output is more compact.
a	double	A real number in format [-] 0xh.hhhh p±dd, where h.hhhh - mantissa in the form of hexadecimal digits, using "abcdef", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
A	double	A real number in format [-] 0xh.hhhh P±dd, where h.hhhh - mantissa in the form of hexadecimal digits, using "ABCDEF", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
s	string	String output

Instead of PrintFormat() you can use [printf\(\)](#).

Example:


```

void OnStart()
{
//--- trade server name
    string server=AccountInfoString(ACCOUNT_SERVER);
//--- account number
    int login=(int)AccountInfoInteger(ACCOUNT_LOGIN);
//--- long value output
    long leverage=AccountInfoInteger(ACCOUNT_LEVERAGE);
    PrintFormat("%s %d: leverage = 1:%I64d",
                server,login,leverage);
//--- account currency
    string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- double value output with 2 digits after the decimal point
    double equity=AccountInfoDouble(ACCOUNT_EQUITY);
    PrintFormat("%s %d: account equity = %.2f %s",
                server,login,equity,currency);
//--- double value output with mandatory output of the +/- sign
    double profit=AccountInfoDouble(ACCOUNT_PROFIT);
    PrintFormat("%s %d: current result for open positions = %.2f %s",
                server,login,profit,currency);
//--- double value output with variable number of digits after the decimal point
    double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
    string format_string=StringFormat("%s: point value = %%.%df",_Digits);
    PrintFormat(format_string,_Symbol,point_value);
//--- int value output
    int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
    PrintFormat("%s: current spread in points = %d ",
                _Symbol,spread);
//--- double value output in the scientific (floating point) format with 17 meaningful
    PrintFormat("DBL_MAX = %.17e",DBL_MAX);
//--- double value output in the scientific (floating point) format with 17 meaningful
    PrintFormat("EMPTY_VALUE = %.17e",EMPTY_VALUE);
//--- output using PrintFormat() with default accuracy
    PrintFormat("PrintFormat(EMPTY_VALUE) = %e",EMPTY_VALUE);
//--- simple output using Print()
    Print("Print(EMPTY_VALUE) = ",EMPTY_VALUE);
/* execution result
MetaQuotes-Demo 1889998: leverage = 1:100
MetaQuotes-Demo 1889998: account equity = 22139.86 USD
MetaQuotes-Demo 1889998: current result for open positions = +174.00 USD
EURUSD: point value = 0.00001
EURUSD: current spread in points = 12
DBL_MAX = 1.79769313486231570e+308
EMPTY_VALUE = 1.79769313486231570e+308
PrintFormat(EMPTY_VALUE) = 1.797693e+308
Print(EMPTY_VALUE) = 1.797693134862316e+308
*/
}

```

See also

[StringFormat](#), [DoubleToString](#), [Real types \(double, float\)](#)

ResetLastError

Sets the value of the predefined variable [_LastError](#) into zero.

```
void ResetLastError();
```

Return Value

No return value.

Note

It should be noted that the [GetLastError\(\)](#) function doesn't zero the `_LastError` variable. Usually the `ResetLastError()` function is called before calling a function, after which an [error](#) appearance is checked.

ResourceCreate

Creates an image resource based on a data set. There are two variants of the function:

Creating a resource based on a file

```
bool ResourceCreate(
    const string      resource_name,      // Resource name
    const string      path                // A relative path to the file
);
```

Creating a resource based on the array of pixels

```
bool ResourceCreate(
    const string      resource_name,      // Resource name
    const uint&       data[],             // Data set as an array
    uint             img_width,          // The width of the image resource
    uint             img_height,         // The height of the image resource
    uint             data_xoffset,       // The horizontal rightward offset of the upper left corner
    uint             data_yoffset,       // The vertical downward offset of the upper left corner
    uint             data_width,         // The total width of the image based on the data array
    ENUM_COLOR_FORMAT color_format       // Color processing method
);
```

Parameters

resource_name

[in] Resource name.

data[][]

[in] A one-dimensional or two-dimensional array for creating a complete image.

img_width

[in] The width of the rectangular image area in pixels to be placed in the resource in the form of an image. It cannot be greater than the *data_width* value.

img_height

[in] The height of the rectangular image area in pixels to be placed in the resource in the form of an image.

data_xoffset

[in] The horizontal rightward offset of the rectangular area of the image.

data_yoffset

[in] The vertical downward offset of the rectangular area of the image.

data_width

[in] Required only for one-dimensional arrays. It denotes the full width of the image from the data set. If *data_width*=0, it is assumed to be equal to *img_width*. For two-dimensional arrays the parameter is ignored and is assumed to be equal to the second dimension of the *data[]* array.

color_format

[in] Color processing method, from a value from the [ENUM_COLOR_FORMAT](#) enumeration.

Return Value

Returns true if successful, otherwise false. To get information about the error call the [GetLastError\(\)](#) function. The following errors may occur:

- 4015 - ERR_RESOURCE_NAME_DUPLICATED (identical names of the dynamic and the [static](#) resource)
- 4016 - ERR_RESOURCE_NOT_FOUND (the resource is not found)
- 4017 - ERR_RESOURCE_UNSUPPORTED_TYPE (this type of resource is not supported)
- 4018 - ERR_RESOURCE_NAME_IS_TOO_LONG (the name of the resource is too long)

Note

If the second version of the function is called for creating the same resource with different width, height and shift parameters, it does not create a new resource, but simply updates the existing one.

The first version of the function is used for uploading images and sounds from files, and the second version is used only for the dynamic creation of images.

Images must be in the BMP format with a color depth of 24 or 32 bits. Sounds can only be in the WAV format. The size of the resource should not exceed 16 Mb.

ENUM_COLOR_FORMAT

Identifier	Description
COLOR_FORMAT_XRGB_NOALPHA	The component of the alpha channel is ignored
COLOR_FORMAT_ARGB_RAW	Color components are not handled by the terminal (must be correctly set by the user)
COLOR_FORMAT_ARGB_NORMALIZE	Color components are handled by the terminal

See also

[Resources](#), [ObjectCreate\(\)](#), [ObjectSetString\(\)](#), [OBJPROP_BMPFILE](#)

ResourceFree

The function deletes [dynamically created resource](#) (freeing the memory allocated for it).

```
bool ResourceFree(  
    const string resource_name    // resource name  
);
```

Parameters

resource_name

[in] [Resource](#) name should start with "::".

Return Value

True if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

Note

ResourceFree() allows mql5 application developers to manage memory consumption when actively working with resources. [Graphical objects](#) bound to the resource being deleted from the memory will be displayed correctly after its deletion. However, newly created graphical objects ([OBJ_BITMAP](#) and [OBJ_BITMAP_LABEL](#)) will not be able to use the deleted resource.

The function deletes only dynamic resources created by the program.

See also

[Resources](#), [ObjectCreate\(\)](#), [PlaySound\(\)](#), [ObjectSetString\(\)](#), [OBJPROP_BMPFILE](#)

ResourceReadImage

The function reads data from the graphical resource [created by ResourceCreate\(\) function](#) or [saved in EX5 file during compilation](#).

```
bool ResourceReadImage (
    const string      resource_name,      // graphical resource name for reading
    uint&             data[],             // array for receiving data from the resource
    uint&             width,              // for receiving the image width in the resource
    uint&             height,             // for receiving the image height in the resource
);
```

Parameters

resource_name

[in] Name of the graphical resource containing an image. To gain access to its own resources, the name is used in brief form "::resourcename". If we download a resource from a compiled EX5 file, the full name should be used with the path relative to MQL5 directory, file and resource names - "path\\filename.ex5::resourcename".

data[][]

[in] One- or two-dimensional array for receiving data from the graphical resource.

img_width

[out] Graphical resource image width in pixels.

img_height

[out] Graphical resource image height in pixels.

Return Value

true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

Note

If *data[]* array is then to be used for [creating a graphical resource](#), [COLOR_FORMAT_ARGB_NORMALIZE](#) or [COLOR_FORMAT_XRGB_NOALPHA](#) color formats should be used.

If *data[]* array is two-dimensional and its second dimension is less than X(width) graphical resource size, ResourceReadImage() function returns false and reading is not performed. But if the resource exists, actual image size is returned to width and height parameters. This will allow making another attempt to receive data from the resource.

See also

[Resource](#), [ObjectCreate\(\)](#), [ObjectSetString\(\)](#), [OBJPROP_BITMAP](#)

ResourceSave

Saves a resource into the specified file.

```
bool ResourceSave(  
    const string resource_name    // Resource name  
    const string file_name       // File name  
);
```

Parameters

resource_name

[in] The name of the resource, must start with "::".

file_name

[in] The name of the file relative to MQL5\Files.

Return Value

true - in case of success, otherwise false. For the error information call [GetLastError\(\)](#).

Note

The function always overwrites a file and creates all the required intermediate directories in the file name if necessary.

See also

[Resources](#), [ObjectCreate\(\)](#), [PlaySound\(\)](#), [ObjectSetString\(\)](#), [OBJPROP_BMPFILE](#)

SetUserError

Sets the predefined variable [LastError](#) into the value equal to [ERR_USER_ERROR_FIRST](#) + user_error

```
void SetUserError(  
    ushort user_error,    // error number  
);
```

Parameters

user_error

[in] [Error](#) number set by a user.

Return Value

No return value.

Note

After an error has been set using the SetUserError(user_error) function, [GetLastError\(\)](#) returns value equal to [ERR_USER_ERROR_FIRST](#) + user_error.

Example:

```
void OnStart()  
{  
    //--- set error number 65537=(ERR_USER_ERROR_FIRST +1)  
    SetUserError(1);  
    //--- get last error code  
    Print("GetLastError = ",GetLastError());  
    /*  
    Result  
    GetLastError = 65537  
    */  
}
```


SendFTP

Sends a file at the address, specified in the setting window of the "FTP" tab.

```
bool SendFTP (
    string filename,           // file to be send by ftp
    string ftp_path=NULL      // ftp catalog
);
```

Parameters

filename

[in] Name of sent file.

ftp_path=NULL

[in] FTP catalog. If a directory is not specified, directory described in settings is used.

Return Value

In case of failure returns 'false'.

Note

Sent file must be located in the folder *terminal_directory\MQL5\files* or its subfolders. Sending isn't performed if FTP address and/or access password are not specified in settings.

SendFTP() function does not work in the [Strategy Tester](#).

SendNotification

Sends push notifications to the mobile terminals, whose MetaQuotes IDs are specified in the "Notifications" tab.

```
bool SendNotification(  
    string text           // Text of the notification  
);
```

Parameters

text

[in] The text of the notification. The message length should not exceed 255 characters.

Return Value

true if a notification has been successfully sent from the terminal; in case of failure returns false. When checking after a failed push of notification, [GetLastError \(\)](#) may return one of the following errors:

- 4515 - ERR_NOTIFICATION_SEND_FAILED,
- 4516 - ERR_NOTIFICATION_WRONG_PARAMETER,
- 4517 - ERR_NOTIFICATION_WRONG_SETTINGS,
- 4518 - ERR_NOTIFICATION_TOO_FREQUENT.

Note

Strict use restrictions are set for the SendNotification() function: no more than 2 calls per second and not more than 10 calls per minute. Monitoring the frequency of use is dynamic. The function can be disabled in case of the restriction violation.

SendNotification() function does not work in the [Strategy Tester](#).

SendMail

Sends an email at the address specified in the settings window of the "Email" tab.

```
bool SendMail(  
    string subject,      // header  
    string some_text     // email text  
);
```

Parameters

subject

[in] Email header.

some_text

[in] Email body.

Return Value

true - if an email is put into the send queue, otherwise - false.

Note

Sending can be prohibited in settings, email address can be omitted as well. For the error information call [GetLastError\(\)](#).

SendMail() function does not work in the [Strategy Tester](#).

Sleep

The function suspends execution of the current Expert Advisor or script within a specified interval.

```
void Sleep(  
    int milliseconds    // interval  
);
```

Parameters

milliseconds

[in] Delay interval in milliseconds.

Return Value

No return value.

Note

The Sleep() function can't be called for custom indicators, because indicators are executed in the interface thread and must not slow down it. The function has the built-in check of EA halt flag every 0.1 seconds.

TerminalClose

The function commands the terminal to complete operation.

```
bool TerminalClose(
    int ret_code    // closing code of the client terminal
);
```

Parameters

ret_code

[in] Return code, returned by the process of the client terminal at the operation completion.

Return Value

The function returns true on success, otherwise - false.

Note

The TerminalClose() function does not stop the terminal immediately, it just commands the terminal to complete its operation.

The code of an Expert Advisor that called TerminalClose() must have all arrangements for the immediate completion (e.g. all previously opened files must be closed in the normal mode). Call of this function must be followed by the [return operator](#).

The *ret_code* parameter allows indicating the necessary return code for analyzing reasons of the program termination of the terminal operation when starting it from the command prompt.

Example:

```
//--- input parameters
input int  tiks_before=500; // number of ticks till termination
input int  pips_to_go=15;   // distance in pips
input int  seconds_st=50;   // number of seconds given to the Expert Advisor
//--- globals
datetime   launch_time;
int        tick_counter=0;
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //---
    Print(__FUNCTION__, " reason code = ", reason);
    Comment("");
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
    static double first_bid=0.0;
```



```

MqlTick      tick;
double       distance;
//---
SymbolInfoTick(_Symbol,tick);
tick_counter++;
if(first_bid==0.0)
{
    launch_time=tick.time;
    first_bid=tick.bid;
    Print("first_bid =",first_bid);
    return;
}
//--- price distance in pips
distance=(tick.bid-first_bid)/_Point;
//--- show a notification to track the EA operation
string comm="From the moment of start:\r\n\x25CF elapsed seconds: "+
            IntegerToString(tick.time-launch_time)+" ;"+
            "\r\n\x25CF ticks received: "+(string)tick_counter+" ;"+
            "\r\n\x25CF price went in points: "+StringFormat("%G",distance);
Comment(comm);
//--- section for checking condition to close the terminal
if(tick_counter>=tiks_before)
    TerminalClose(0);    // exit by tick counter
if(distance>pips_to_go)
    TerminalClose(1);    // go up by the number of pips equal to pips_to_go
if(distance<-pips_to_go)
    TerminalClose(-1);   // go down by the number of pips equal to pips_to_go
if(tick.time-launch_time>seconds_st)
    TerminalClose(100);  // termination by timeout
//---
}

```

See also

[Program running](#), [Execution errors](#), [Reasons for deinitialization](#)

TesterStatistics

The function returns the value of the specified statistical parameter calculated based on testing results.

```
double TesterStatistics(  
    ENUM_STATISTICS statistic_id    // ID  
);
```

Parameters

statistic_id

[in] The ID of the statistical parameter from the [ENUM_STATISTICS](#) enumeration.

Return Value

The value of the statistical parameter from testing results.

Note

The function can be called inside [OnTester\(\)](#) or [OnDeinit\(\)](#) in the tester. In other cases the result is undefined.

TesterWithdrawal

The special function to emulate the operation of money withdrawal in the process of testing. Can be used in some asset management systems.

```
bool TesterWithdrawal(  
    double money    // the sum to withdraw  
);
```

Parameters

money

[in] The sum of money that we need to withdraw (in the deposit currency).

Return Value

If successful, returns true, otherwise - false.

TranslateKey

Returns a Unicode character by a virtual key code considering the current input language and the status of control keys.

```
short TranslateKey(  
    int key_code    // key code for receiving a Unicode character  
);
```

Parameters

key_code
[in] Key code.

Return Value

Unicode character in case of a successful conversion. The function returns -1 in case of an error.

Note

The function uses [ToUnicodeEx](#) to convert keys pressed by a user into Unicode characters. An error may occur in case ToUnicodeEx is not triggered - for example, when trying to receive the SHIFT key character.

Example:

```
void OnChartEvent(const int id,const long& lparam,const double& dparam,const string& s  
{  
    if(id==CHARTEVENT_KEYDOWN)  
    {  
        short sym=TranslateKey((int)lparam);  
        //--- if the entered character is successfully converted to Unicode  
        if(sym>0)  
            Print(sym,"",ShortToString(sym,""));  
        else  
            Print("Error in TranslateKey for key=",lparam);  
    }  
}
```

See also

[Client terminal events](#), [OnChartEvent](#)

WebRequest

The function sends an HTTP request to a specified server. The function has two versions:

1. **Sending simple requests** of type "key=value" using the header Content-Type: application/x-www-form-urlencoded.

```
int WebRequest(
    const string    method,           // HTTP method
    const string    url,              // URL
    const string    cookie,           // cookie
    const string    referer,          // referer
    int             timeout,          // timeout
    const char      &data[],          // the array of the HTTP message body
    int             data_size,        // data[] array size in bytes
    char            &result[],        // an array containing server response data
    string          &result_headers // headers of server response
);
```

2. **Sending a request of any type** specifying the custom set of headers for a more flexible interaction with various Web services.

```
int WebRequest(
    const string    method,           // HTTP method
    const string    url,              // URL
    const string    headers,          // headers
    int             timeout,          // timeout
    const char      &data[],          // the array of the HTTP message body
    char            &result[],        // an array containing server response data
    string          &result_headers // headers of server response
);
```

Parameters

method

[in] HTTP method.

url

[in] URL.

headers

[in] Request headers of type "key: value", separated by a line break "\r\n".

cookie

[in] Cookie value.

referer

[in] Value of the Referer header of the HTTP request.

timeout

[in] Timeout in milliseconds.

data[]

[in] Data array of the HTTP message body.

data_size

[in] Size of the *data[]* array.

result[]

[out] An array containing server response data.

result_headers

[out] Server response headers.

Return Value

HTTP server response code or -1 for an error.

Note

To use the `WebRequest()` function, add the addresses of the required servers in the list of allowed URLs in the "Expert Advisors" tab of the "Options" window. Server port is automatically selected on the basis of the specified protocol - 80 for "http://" and 443 for "https://".

The `WebRequest()` function is synchronous, which means it breaks the program execution and waits for the response from the requested server. Since the delays in receiving a response can be large, the function is not available for calls from the indicators, because indicators run in a common thread shared by all indicators and charts on one symbol. Indicator performance delay on one of the charts of a symbol may stop updating of all charts of the same symbol.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If you try to call the function from an indicator, [GetLastError\(\)](#) will return error 4014 - "Function is not allowed for call".

`WebRequest()` cannot be executed in the [Strategy Tester](#).

An example of using the first version of the `WebRequest()` function:

```
void OnStart()
{
    string cookie=NULL,headers;
    char post[],result[];
    int res;
    //--- to enable access to the server, you should add URL "https://www.google.com/finance"
    //--- in the list of allowed URLs (Main Menu->Tools->Options, tab "Expert Advisors"):
    string google_url="https://www.google.com/finance";
    //--- Reset the last error code
    ResetLastError();
    //--- Loading a html page from Google Finance
    int timeout=5000; //--- Timeout below 1000 (1 sec.) is not enough for slow Internet
    res=WebRequest("GET",google_url,cookie,NULL,timeout,post,0,result,headers);
    //--- Checking errors
    if(res==-1)
    {
        Print("Error in WebRequest. Error code =",GetLastError());
    }
}
```



```

    //--- Perhaps the URL is not listed, display a message about the necessity to add it
    MessageBox("Add the address '"+google_url+"' in the list of allowed URLs on tab
}
else
{
    //--- Load successfully
    PrintFormat("The file has been successfully loaded, File size =%d bytes.",ArraySize(result));
    //--- Save the data to a file
    int filehandle=FileOpen("GoogleFinance.htm",FILE_WRITE|FILE_BIN);
    //--- Checking errors
    if(filehandle!=INVALID_HANDLE)
    {
        //--- Save the contents of the result[] array to a file
        FileWriteArray(filehandle,result,0,ArraySize(result));
        //--- Close the file
        FileClose(filehandle);
    }
    else Print("Error in FileOpen. Error code=",GetLastError());
}
}

```

An example of using the second version of the WebRequest() function:

```

#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
#property script_show_inputs
#property description "Sample script posting a user message "
#property description "on the wall on mql5.com"

input string InpLogin    ="";           //Your MQL5.com account
input string InpPassword="";           //Your account password
input string InpFileName="EURUSDM5.png"; //An image in folder MQL5/Files/
input string InpFileType="image/png";   //Correct mime type of the image
//+-----+
//| Posting a message with an image on the wall at mql5.com |
//+-----+

bool PostToNewsFeed(string login,string password,string text,string filename,string filetype)
{
    int res; // To receive the operation execution result
    char data[]; // Data array to send POST requests
    char file[]; // Read the image here
    string str="Login="+login+"&Password="+password;
    string auth,sep="-----Jyecslin9mp8RdKV"; // multipart data separator
    //--- A file is available, try to read it
    if(filename!=NULL && filename!="")
    {
        res=FileOpen(filename,FILE_READ|FILE_BIN);
    }
}

```



```

    if(res<0)
    {
        Print("Error opening the file \""+filename+"\"");
        return(false);
    }
    //--- Read file data
    if(FileReadArray(res,file)!=FileSize(res))
    {
        FileClose(res);
        Print("Error reading the file \""+filename+"\"");
        return(false);
    }
    //---
    FileClose(res);
}

//--- Create the body of the POST request for authorization
ArrayResize(data,StringToCharArray(str,data,0,WHOLE_ARRAY,CP_UTF8)-1);
//--- Resetting error code
ResetLastError();
//--- Authorization request
res=WebRequest("POST","https://www.mql5.com/en/auth_login",NULL,0,data,data,str);
//--- If authorization failed
if(res!=200)
{
    Print("Authorization error #"+(string)res+", GetLastError="+ (string)GetLastError());
    return(false);
}

//--- Read the authorization cookie from the server response header
res=StringFind(str,"Set-Cookie: auth=");
//--- If cookie not found, return an error
if(res<0)
{
    Print("Error, authorization data not found in the server response (check login/");
    return(false);
}

//--- Remember the authorization data and form the header for further requests
auth=StringSubstr(str,res+12);
auth="Cookie: "+StringSubstr(auth,0,StringFind(auth,";")+1)+"\r\n";
//--- If there is a data file, send it to the server
if(ArraySize(file)!=0)
{
    //--- Form the request body
    str="--"+sep+"\r\n";
    str+="Content-Disposition: form-data; name=\"attachedFile_imagesLoader\"; filename=";
    str+="Content-Type: "+filetype+"\r\n\r\n";
    res =StringToCharArray(str,data);
    res+=ArrayCopy(data,file,res-1,0);
    res+=StringToCharArray("\r\n--"+sep+"--\r\n",data,res-1);
    ArrayResize(data,ArraySize(data)-1);
}

```



```

//--- Form the request header
str=auth+"Content-Type: multipart/form-data; boundary="+sep+"\r\n";
//--- Reset error code
ResetLastError();
//--- Request to send an image file to the server
res=WebRequest("POST","https://www.mql5.com/upload_file",str,0,data,data,str);
//--- check the request result
if(res!=200)
{
    Print("Error sending a file to the server #"+(string)res+", GetLastError="+GetLastError());
    return(false);
}
//--- Receive a link to the image uploaded to the server
str=CharArrayToString(data);
if(StringFind(str,{"Url":"":""})==0)
{
    res=StringFind(str,"\"",8);
    filename=StringSubstr(str,8,res-8);
    //--- If file uploading fails, an empty link will be returned
    if(filename=="")
    {
        Print("File sending to server failed");
        return(false);
    }
}
}

//--- Create the body of a request to post an image on the server
str="--"+sep+"\r\n";
str+="Content-Disposition: form-data; name=\"content\""\r\n\r\n";
str+=text+"\r\n";
//--- The languages in which the post will be available on mql5.com
str="--"+sep+"\r\n";
str+="Content-Disposition: form-data; name=\"AllLanguages\""\r\n\r\n";
str+="on\r\n";
//--- If the picture has been uploaded on the server, pass its link
if(ArraySize(file)!=0)
{
    str="--"+sep+"\r\n";
    str+="Content-Disposition: form-data; name=\"attachedImage_0\""\r\n\r\n";
    str+=filename+"\r\n";
}
//--- The final string of the multipart request
str="--"+sep+"--\r\n";
//--- Out the body of the POST request together in one string
StringToCharArray(str,data,0,WHOLE_ARRAY,CP_UTF8);
ArrayResize(data,ArraySize(data)-1);
//--- Prepare the request header
str=auth+"Content-Type: multipart/form-data; boundary="+sep+"\r\n";
//--- Request to post a message on the user wall at mql5.com

```



```

    res=WebRequest("POST","https://www.mql5.com/ru/users/"+login+"/wall",str,0,data,data);
//--- Return true for successful execution
    return(res==200);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- Post a message on mql5.com, including an image, the path to which is taken from
    PostToNewsFeed(InpLogin,InpPassword,"Checking the expanded version of WebRequest\r\n
        "(This message has been posted by the WebRequest.mq5 script)",InpFile);
}
//+-----+

```


ZeroMemory

The function resets a variable passed to it by reference.

```
void ZeroMemory(  
    void & variable    // reset variable  
);
```

Parameters

variable

[in] [out] Variable passed by reference, you want to reset (initialize by zero values).

Return Value

No return value.

Note

If the function parameter is a string, the call will be equivalent to indicating NULL as its value. For simple types and their arrays, as well as for structures/classes consisting of such types, this is a simple reset.

For objects containing strings and dynamic arrays, ZeroMemory() is called for each element.

For any arrays not protected by the const modifier, this is the zeroing of all elements.

For arrays of complex objects, ZeroMemory() is called for each element.

ZeroMemory() can't be applied to classes with protected [members](#) or [inheritance](#).

Group of Functions for Working with Arrays

[Arrays](#) are allowed to be maximum four-dimensional. Each dimension is indexed from 0 to *dimension_size-1*. In a particular case of a one-dimensional array of 50 elements, calling of the first element will appear as `array[0]`, of the last one - as `array[49]`.

Function	Action
ArrayBsearch	Returns index of the first found element in the first array dimension
ArrayCopy	Copies one array into another
ArrayCompare	Returns the result of comparing two arrays of simple types or custom structures without complex objects
ArrayFree	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.
ArrayGetAsSeries	Checks direction of array indexing
ArrayInitialize	Sets all elements of a numeric array into a single value
ArrayFill	Fills an array with the specified value
ArrayIsSeries	Checks whether an array is a timeseries
ArrayIsDynamic	Checks whether an array is dynamic
ArrayMaximum	Search for an element with the maximal value
ArrayMinimum	Search for an element with the minimal value
ArrayPrint	Prints an array of a simple type or a simple structure into journal
ArrayRange	Returns the number of elements in the specified dimension of the array
ArrayResize	Sets the new size in the first dimension of the array
ArraySetAsSeries	Sets the direction of array indexing
ArraySize	Returns the number of elements in the array
ArraySort	Sorting of numeric arrays by the first dimension

ArrayBsearch

Searches for a specified value in a multidimensional numeric array [sorted](#) ascending. Search is performed through the elements of the first dimension.

For searching in an array of double type

```
int ArrayBsearch(  
    const double&    array[],    // array for search  
    double          value        // what is searched for  
);
```

For searching in an array of float type

```
int ArrayBsearch(  
    const float&     array[],    // array for search  
    float           value        // what is searched for  
);
```

For searching in an array of long type

```
int ArrayBsearch(  
    const long&      array[],    // array for search  
    long            value        // what is searched for  
);
```

For searching in an array of int type

```
int ArrayBsearch(  
    const int&       array[],    // array for search  
    int             value        // what is searched for  
);
```

For searching in an array of short type

```
int ArrayBsearch(  
    const short&     array[],    // array for search  
    short           value        // what is searched for  
);
```

For searching in an array of char type

```
int ArrayBsearch(  
    const char&      array[],    // array for search  
    char            value        // what is searched for  
);
```

Parameters

array[]

[in] Numeric array for search.

value

[in] Value for search.

Return Value

The function returns index of a found element. If the wanted value isn't found, the function returns the index of an element nearest in value.

Note

Binary search processes only sorted arrays. To sort numeric arrays use the [ArraySort\(\)](#) function.

Example:

```
#property description "Script based on RSI indicator data displays"
#property description "how often the market was in"
#property description "overbought and oversold areas in the specified time interval."
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input int                InpMAPeriod=14;                // Moving average period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;    // Price type
input double             InpOversoldValue=30.0;          // Oversold level
input double             InpOverboughtValue=70.0;        // Overbought level
input datetime           InpDateStart=D'2012.01.01 00:00'; // Analysis start date
input datetime           InpDateFinish=D'2013.01.01 00:00'; // Analysis finish date
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    double rsi_buff[]; // array of the indicator values
    int     size=0;     // array size
//--- receive RSI indicator handle
    ResetLastError();
    int rsi_handle=iRSI(Symbol(),Period(),InpMAPeriod,InpAppliedPrice);
    if(rsi_handle==INVALID_HANDLE)
    {
        //--- failed to receive the indicator handle
        PrintFormat("Indicator handle receiving error. Error code = %d",GetLastError());
        return;
    }
//--- being in the loop, until the indicator calculates all its values
    while(BarsCalculated(rsi_handle)==-1)
    {
        //--- exit if the indicator has forcedly completed the script's operation
        if(IsStopped())
            return;
        //--- a pause to allow the indicator to calculate all its values
        Sleep(10);
    }
//--- copy the indicator values for a certain period of time
```



```

ResetLastError();
if(CopyBuffer(rsi_handle,0,InpDateStart,InpDateFinish,rsi_buff)==-1)
{
    PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError()
    return;
}
//--- receive the array size
size=ArraySize(rsi_buff);
//--- sort out the array
ArraySort(rsi_buff);
//--- find out the time (in percentage terms) the market was in the oversold area
double ovs=(double)ArrayBsearch(rsi_buff,InpOversoldValue)*100/(double)size;
//--- find out the time (in percentage terms) the market was in the overbought area
double ovb=(double)(size-ArrayBsearch(rsi_buff,InpOverboughtValue))*100/(double)size;
//--- form the strings for displaying the data
string str="From "+TimeToString(InpDateStart,TIME_DATE)+" to "
        +TimeToString(InpDateFinish,TIME_DATE)+" the market was:";
string str_ovb="in overbought area "+DoubleToString(ovb,2)+"% of time";
string str_ovs="in oversold area "+DoubleToString(ovs,2)+"% of time";
//--- display the data on the chart
CreateLabel("top",5,60,str,clrDodgerBlue);
CreateLabel("overbought",5,35,str_ovb,clrDodgerBlue);
CreateLabel("oversold",5,10,str_ovs,clrDodgerBlue);
//--- redraw the chart
ChartRedraw(0);
//--- pause
Sleep(10000);
}
//+-----+
//| Display comment in the bottom left corner of the chart |
//+-----+
void CreateLabel(const string name,const int x,const int y,
                const string str,const color clr)
{
    //--- create the label
    ObjectCreate(0,name,OBJ_LABEL,0,0,0);
    //--- bind the label to the bottom left corner
    ObjectSetInteger(0,name,OBJPROP_CORNER,CORNER_LEFT_LOWER);
    //--- change position of the anchor point
    ObjectSetInteger(0,name,OBJPROP_ANCHOR,ANCHOR_LEFT_LOWER);
    //--- distance from the anchor point in X-direction
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
    //--- distance from the anchor point in Y-direction
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
    //--- label text
    ObjectSetString(0,name,OBJPROP_TEXT,str);
    //--- text color
    ObjectSetInteger(0,name,OBJPROP_COLOR,clr);
    //--- text size

```



```
ObjectSetInteger(0,name,OBJPROP_FONTSIZE,12);  
}
```


ArrayCopy

It copies an array into another one.

```
int ArrayCopy(
    void&      dst_array[],      // destination array
    const void& src_array[],      // source array
    int        dst_start=0,      // index starting from which write into destination array
    int        src_start=0,      // first index of a source array
    int        count=WHOLE_ARRAY // number of elements
);
```

Parameters

dst_array[]

[out] Destination array

src_array[]

[in] Source array

dst_start=0

[in] Starting index from the destination array. By default, start index is 0.

src_start=0

[in] Starting index for the source array. By default, start index is 0.

count=WHOLE_ARRAY

[in] Number of elements that should be copied. By default, the whole array is copied (count=[WHOLE_ARRAY](#)).

Return Value

It returns the number of copied elements.

Note

If $\text{count} < 0$ or $\text{count} > \text{src_size} - \text{src_start}$, all the remaining array part is copied. Arrays are copied from left to right. For series arrays, the starting position is correctly defined adjusted for copying from left to right. If an array is copied to itself, the result is undefined.

If arrays are of different types, during copying it tries to transform each element of a source array into the type of the destination array. A string array can be copied into a string array only. Array of [classes and structures](#) containing objects that require initialization aren't copied. An array of structures can be copied into an array of the same type only.

For dynamic arrays with indexing as in [timeseries](#), the size of a destination array is automatically increased to the amount of copied data (if the latter exceeds the array size). The destination array size is not decreased automatically.

Example:

```
#property description "The indicator highlights the candlesticks that are local"
#property description "highs and lows. Interval length for finding"
#property description "extreme values should be found using an input parameters."
```



```

//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 1
//---- plot
#property indicator_label1 "Extremums"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_color1 clrLightSteelBlue,clrRed,clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input int InpNum=4; // Half-interval length
//--- indicator buffers
double ExtOpen[];
double ExtHigh[];
double ExtLow[];
double ExtClose[];
double ExtColor[];
//--- global variables
int ExtStart=0; // index of the first candlestick that is not an extremum
int ExtCount=0; // number of non-extremums in the interval
//+-----+
//| Filling out non-extremum candlesticks |
//+-----+
void FillCandles(const double &open[],const double &high[],
                 const double &low[],const double &close[])
{
//--- fill out the candlesticks
    ArrayCopy(ExtOpen,open,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtHigh,high,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtLow,low,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtClose,close,ExtStart,ExtStart,ExtCount);
}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ExtOpen);
    SetIndexBuffer(1,ExtHigh);
    SetIndexBuffer(2,ExtLow);
    SetIndexBuffer(3,ExtClose);
    SetIndexBuffer(4,ExtColor,INDICATOR_COLOR_INDEX);
//--- specify the value, which is not displayed
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- specify the names of indicator buffers for displaying in the data window

```



```

    PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- set straight indexing in time series
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);
//--- variable of the bar calculation start
    int start=prev_calculated;
//--- calculation is not performed for the first InpNum*2 bars
    if(start==0)
    {
        start+=InpNum*2;
        ExtStart=0;
        ExtCount=0;
    }
//--- if the bar has just formed, check the next potential extremum
    if(rates_total-start==1)
        start--;
//--- bar index to be checked for the extremum
    int ext;
//--- indicator value calculation loop
    for(int i=start;i<rates_total-1;i++)
    {
        //--- initially on i bar without drawing
        ExtOpen[i]=0;
        ExtHigh[i]=0;
        ExtLow[i]=0;
        ExtClose[i]=0;
        //--- extremum index for check
        ext=i-InpNum;
        //--- check for the local maximum
        if(IsMax(high,ext))

```



```

    {
        //--- highlight an extremum candlestick
        ExtOpen[ext]=open[ext];
        ExtHigh[ext]=high[ext];
        ExtLow[ext]=low[ext];
        ExtClose[ext]=close[ext];
        ExtColor[ext]=1;
        //--- highlight other candles up to the extremum with a neutral color
        FillCandles(open,high,low,close);
        //--- change the variable colors
        ExtStart=ext+1;
        ExtCount=0;
        //--- pass to the next iteration
        continue;
    }
    //--- check for the local minimum
    if(IsMin(low,ext))
    {
        //--- highlight an extremum candlestick
        ExtOpen[ext]=open[ext];
        ExtHigh[ext]=high[ext];
        ExtLow[ext]=low[ext];
        ExtClose[ext]=close[ext];
        ExtColor[ext]=2;
        //--- highlight other candles up to the extremum with a neutral color
        FillCandles(open,high,low,close);
        //--- change variable values
        ExtStart=ext+1;
        ExtCount=0;
        //--- pass to the next iteration
        continue;
    }
    //--- increase the number of non-extremums at the interval
    ExtCount++;
}
//--- return value of prev_calculated for next call
return(rates_total);
}
//+-----+
//| Check if the current array element is a local high |
//+-----+
bool IsMax(const double &price[],const int ind)
{
    //--- interval start variable
    int i=ind-InpNum;
    //--- interval end period
    int finish=ind+InpNum+1;
    //--- check for the first half of the interval
    for(;i<ind;i++)

```



```

        {
            if(price[ind]<=price[i])
                return(false);
        }
    //--- check for the second half of the interval
    for(i=ind+1;i<finish;i++)
    {
        if(price[ind]<=price[i])
            return(false);
    }
    //--- this is an extremum
    return(true);
}

//+-----+
//| Check if the current array element is a local low |
//+-----+
bool IsMin(const double &price[],const int ind)
{
    //--- interval start variable
    int i=ind-InpNum;
    //--- interval end variable
    int finish=ind+InpNum+1;
    //--- check for the first half of the interval
    for(;i<ind;i++)
    {
        if(price[ind]>=price[i])
            return(false);
    }
    //--- check for the second half of the interval
    for(i=ind+1;i<finish;i++)
    {
        if(price[ind]>=price[i])
            return(false);
    }
    //--- this is an extremum
    return(true);
}

```


ArrayCompare

The function returns the result of comparing two arrays of the same type. It can be used to compare arrays of [simple types](#) or custom structures without [complex objects](#), that is the custom structures that do not contain [strings](#), [dynamic arrays](#), classes and other structures with complex objects.

```
int ArrayCompare(
    const void& array1[],           // first array
    const void& array2[],           // second array
    int start1=0,                   // initial offset in the first array
    int start2=0,                   // initial offset in the second array
    int count=WHOLE_ARRAY           // number of elements for comparison
);
```

Parameters

array1[]

[in] First array.

array2[]

[in] Second array.

start1=0

[in] The element's initial index in the first array, from which comparison starts. The default start index - 0.

start2=0

[in] The element's initial index in the second array, from which comparison starts. The default start index - 0.

count=WHOLE_ARRAY

[in] Number of elements to be compared. All elements of both arrays participate in comparison by default (count=[WHOLE_ARRAY](#)).

Return Value

- -1, if array1[] less than array2[]
- 0, if array1[] equal to array2[]
- 1, if array1[] more than array2[]
- -2, if an error occurs due to incompatibility of the types of compared arrays or if start1, start2 or count values lead to falling outside the array.

Note

The function will not return 0 (the arrays will not be considered equal) if the arrays differ in size and count=WHOLE_ARRAY for the case when one array is a faithful subset of another one. In this case, the result of comparing the sizes of that arrays will be returned: -1, if the size of array1[] is less than the size of array2[] , otherwise 1.

ArrayFree

It frees up a buffer of any dynamic array and sets the size of the zero dimension to 0.

```
void ArrayFree(
    void& array[]    // array
);
```

Parameters

array[]
[in] Dynamic array.

Return Value

No return value.

Note

The need for using ArrayFree() function may not appear too often considering that all used memory is freed at once and main work with the arrays comprises the access to the indicator buffers. The sizes of the buffers are automatically managed by the terminal's executive subsystem.

In case it is necessary to manually manage the memory in complex dynamic environment of the application, ArrayFree() function allows users to free the memory occupied by the already unnecessary dynamic array explicitly and immediately.

Example:

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- predefined constants
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+-----+
//| Dialog class for working with memory |
//+-----+
class CMemoryControl : public CAppDialog
{
private:
    //--- array size
    int      m_arr_size;
    //--- arrays
    char      m_arr_char[];
    int      m_arr_int[];
    float     m_arr_float[];
    double    m_arr_double[];
    long      m_arr_long[];
```



```

//--- labels
CLabel          m_lbl_memory_physical;
CLabel          m_lbl_memory_total;
CLabel          m_lbl_memory_available;
CLabel          m_lbl_memory_used;
CLabel          m_lbl_array_size;
CLabel          m_lbl_array_type;
CLabel          m_lbl_error;
CLabel          m_lbl_change_type;
CLabel          m_lbl_add_size;

//--- buttons
CButton         m_button_add;
CButton         m_button_free;

//--- combo boxes
CComboBox       m_combo_box_step;
CComboBox       m_combo_box_type;

//--- current value of the array type from the combo box
int             m_combo_box_type_value;

public:
                CMemoryControl(void);
                ~CMemoryControl(void);

//--- class object creation method
virtual bool    Create(const long chart,const string name,const int subwin,const
//--- handler of chart events
virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
//--- create labels
bool            CreateLabel(CLabel &lbl,const string name,const int x,const int y
//--- create control elements
bool            CreateButton(CButton &button,const string name,const int x,const
bool            CreateComboBoxStep(void);
bool            CreateComboBoxType(void);

//--- event handlers
void            OnClickButtonAdd(void);
void            OnClickButtonFree(void);
void            OnChangeComboBoxType(void);

//--- methods for working with the current array
void            CurrentArrayFree(void);
bool            CurrentArrayAdd(void);

};

//+-----+
//| Free memory of the current array |
//+-----+
void CMemoryControl::CurrentArrayFree(void)
{
//--- reset array size
m_arr_size=0;

```



```

//--- free the array
if(m_combo_box_type_value==0)
    ArrayFree(m_arr_char);
if(m_combo_box_type_value==1)
    ArrayFree(m_arr_int);
if(m_combo_box_type_value==2)
    ArrayFree(m_arr_float);
if(m_combo_box_type_value==3)
    ArrayFree(m_arr_double);
if(m_combo_box_type_value==4)
    ArrayFree(m_arr_long);
}
//+-----+
//| Attempt to add memory for the current array |
//+-----+
bool CMemoryControl::CurrentArrayAdd(void)
{
//--- exit if the size of the used memory exceeds the size of the physical memory
if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TERMINAL_MEMORY_FREE)>1)
    return(false);
//--- attempt to allocate memory according to the current type
if(m_combo_box_type_value==0 && ArrayResize(m_arr_char,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==1 && ArrayResize(m_arr_int,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==2 && ArrayResize(m_arr_float,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==3 && ArrayResize(m_arr_double,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==4 && ArrayResize(m_arr_long,m_arr_size)==-1)
    return(false);
//--- memory allocated
return(true);
}
//+-----+
//| Handling events |
//+-----+
EVENT_MAP_BEGIN(CMemoryControl)
ON_EVENT(ON_CLICK,m_button_add,OnClickButtonAdd)
ON_EVENT(ON_CLICK,m_button_free,OnClickButtonFree)
ON_EVENT(ON_CHANGE,m_combo_box_type,OnChangeComboBoxType)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CMemoryControl::CMemoryControl(void)
{
}
//+-----+

```



```

//| Destructor |
//+-----+
CMemoryControl::~CMemoryControl(void)
{
}
//+-----+
//| Class object creation method |
//+-----+
bool CMemoryControl::Create(const long chart,const string name,const int subwin,
                           const int x1,const int y1,const int x2,const int y2)
{
//--- create base class object
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- prepare strings for labels
    string str_physical="Memory physical = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL);
    string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_TOTAL);
    string str_available="Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE);
    string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED);
//--- create labels
    if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_START+5,str_physical,12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,str_total,12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_START+55,str_available,12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,str_used,12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"Array type = ",12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"Array size = ",12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,clrRed))
        return(false);
    if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_START+185,"Change type",12,clrBlue))
        return(false);
    if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,"Add to array",12,clrBlue))
        return(false);
//--- create control elements
    if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add",12,clrBlue))
        return(false);
    if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Free",12,clrBlue))
        return(false);
    if(!CreateComboBoxType())
        return(false);
    if(!CreateComboBoxStep())
        return(false);
//--- initialize the variable
    m_arr_size=0;

```



```

//--- successful execution
    return(true);
}
//+-----+
//| Create the button                                     |
//+-----+
bool CMemoryControl::CreateButton(CButton &button,const string name,const int x,
                                  const int y,const string str,const int font_size,
                                  const int clr)
{
//--- create the button
    if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
        return(false);
//--- text
    if(!button.Text(str))
        return(false);
//--- font size
    if(!button.FontSize(font_size))
        return(false);
//--- label color
    if(!button.Color(clr))
        return(false);
//--- add the button to the control elements
    if(!Add(button))
        return(false);
//--- successful execution
    return(true);
}
//+-----+
//| Create a combo box for the array size                 |
//+-----+
bool CMemoryControl::CreateComboBoxStep(void)
{
//--- create the combo box
    if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START+100,Y_START)
        return(false);
//--- add elements to the combo box
    if(!m_combo_box_step.ItemAdd("100 000",100000))
        return(false);
    if(!m_combo_box_step.ItemAdd("1 000 000",1000000))
        return(false);
    if(!m_combo_box_step.ItemAdd("10 000 000",10000000))
        return(false);
    if(!m_combo_box_step.ItemAdd("100 000 000",100000000))
        return(false);
//--- set the current combo box element
    if(!m_combo_box_step.SelectByValue(1000000))
        return(false);
//--- add the combo box to control elements

```



```

    if(!Add(m_combo_box_step))
        return(false);
//--- successful execution
    return(true);
}
//+-----+
//| Create a combo box for the array type |
//+-----+
bool CMemoryControl::CreateComboBoxType(void)
{
//--- create the combo box
    if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START+100,Y_START))
        return(false);
//--- add elements to the combo box
    if(!m_combo_box_type.ItemAdd("char",0))
        return(false);
    if(!m_combo_box_type.ItemAdd("int",1))
        return(false);
    if(!m_combo_box_type.ItemAdd("float",2))
        return(false);
    if(!m_combo_box_type.ItemAdd("double",3))
        return(false);
    if(!m_combo_box_type.ItemAdd("long",4))
        return(false);
//--- set the current combo box element
    if(!m_combo_box_type.SelectByValue(3))
        return(false);
//--- store the current combo box element
    m_combo_box_type_value=3;
//--- add the combo box to control elements
    if(!Add(m_combo_box_type))
        return(false);
//--- successful execution
    return(true);
}
//+-----+
//| Create a label |
//+-----+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x,
                                const int y,const string str,const int font_size,
                                const int clr)
{
//--- create a label
    if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
        return(false);
//--- text
    if(!lbl.Text(str))
        return(false);
//--- font size

```



```

        if(!lbl.FontSize(font_size))
            return(false);
//--- color
        if(!lbl.Color(clr))
            return(false);
//--- add the label to control elements
        if(!Add(lbl))
            return(false);
//--- succeed
        return(true);
    }
//+-----+
//| Handler of clicking "Add" button event |
//+-----+
void CMemoryControl::OnClickButtonAdd(void)
{
//--- increase the array size
    m_arr_size+=(int)m_combo_box_step.Value();
//--- attempt to allocate memory for the current array
    if(CurrentArrayAdd())
    {
        //--- memory allocated, display the current status on the screen
        m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE));
        m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED));
        m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
        m_lbl_error.Text("");
    }
    else
    {
        //--- failed to allocate memory, display the error message
        m_lbl_error.Text("Array is too large, error!");
        //--- return the previous array size
        m_arr_size-=(int)m_combo_box_step.Value();
    }
}
//+-----+
//| Handler of clicking "Free" button event |
//+-----+
void CMemoryControl::OnClickButtonFree(void)
{
//--- free the memory of the current array
    CurrentArrayFree();
//--- display the current status on the screen
    m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE));
    m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED));
    m_lbl_array_size.Text("Array size = 0");
    m_lbl_error.Text("");
}
//+-----+

```



```

//| Handler of the combo box change event |
//+-----+
void CMemoryControl::OnChangeComboBoxType(void)
{
//--- check if the array's type has changed
    if(m_combo_box_type.Value()!=m_combo_box_type_value)
    {
        //--- free the memory of the current array
        OnClickButtonFree();
        //--- work with another array type
        m_combo_box_type_value=(int)m_combo_box_type.Value();
        //--- display the new array type on the screen
        if(m_combo_box_type_value==0)
            m_lbl_array_type.Text("Array type = char");
        if(m_combo_box_type_value==1)
            m_lbl_array_type.Text("Array type = int");
        if(m_combo_box_type_value==2)
            m_lbl_array_type.Text("Array type = float");
        if(m_combo_box_type_value==3)
            m_lbl_array_type.Text("Array type = double");
        if(m_combo_box_type_value==4)
            m_lbl_array_type.Text("Array type = long");
    }
}

//--- CMemoryControl class object
CMemoryControl ExtDialog;

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create the dialog
    if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
        return(INIT_FAILED);
    //--- launch
    ExtDialog.Run();
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //---
    ExtDialog.Destroy(reason);
}

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

```



```
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
    ExtDialog.ChartEvent(id, lparam, dparam, sparam);
}
```


ArrayGetAsSeries

It checks direction of an array index.

```
bool ArrayGetAsSeries(
    const void& array[] // array for checking
);
```

Parameters

array

[in] Checked array.

Return Value

Returns [true](#), if the specified array has the AS_SERIES flag set, i.e. access to the array is performed back to front as in timeseries. A [timeseries](#) differs from a usual array in that the indexing of timeseries elements is performed from its end to beginning (from the newest data to old).

Note

To check whether an array belongs to timeseries, use the [ArrayIsSeries\(\)](#) function. Arrays of price data passed as input parameters into the [OnCalculate\(\)](#) function do not obligatorily have the indexing direction the same as in timeseries. The necessary indexing direction can be set using the [ArraySetAsSeries\(\)](#) function.

Example:

```
#property description "Indicator calculates absolute values of the difference between"
#property description "Open and Close or High and Low prices displaying them in a sepa"
#property description "as a histogram."
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_style1 STYLE_SOLID
#property indicator_width1 3
//--- input parameters
input bool InpAsSeries=true; // Indexing direction in the indicator buffer
input bool InpPrices=true; // Calculation prices (true - Open,Close; false - High,Lo
//--- indicator buffer
double ExtBuffer[];
//+-----+
//| Calculating indicator values |
//+-----+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                        const double &first[],const double &second[],double &buffer[])
{
    //--- start variable for calculation of bars
    int start=prev_calculated;
```



```

//--- work at the last bar if the indicator values have already been calculated at the
    if(prev_calculated>0)
        start--;
//--- define indexing direction in arrays
    bool as_series_first=ArrayGetAsSeries(first);
    bool as_series_second=ArrayGetAsSeries(second);
    bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- replace indexing direction with direct one if necessary
    if(as_series_first)
        ArraySetAsSeries(first,false);
    if(as_series_second)
        ArraySetAsSeries(second,false);
    if(as_series_buffer)
        ArraySetAsSeries(buffer,false);
//--- calculate indicator values
    for(int i=start;i<rates_total;i++)
        buffer[i]=MathAbs(first[i]-second[i]);
}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- bind indicator buffers
    SetIndexBuffer(0,ExtBuffer);
//--- set indexing element in the indicator buffer
    ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- check for what prices the indicator is calculated
    if(InpPrices)
    {
        //--- Open and Close prices
        PlotIndexSetString(0,PLOT_LABEL,"BodySize");
        //--- set the indicator color
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrOrange);
    }
    else
    {
        //--- High and Low prices
        PlotIndexSetString(0,PLOT_LABEL,"ShadowSize");
        //--- set the indicator color
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrDodgerBlue);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,

```



```
        const int prev_calculated,\n        const datetime &time[],\n        const double &open[],\n        const double &high[],\n        const double &low[],\n        const double &close[],\n        const long &tick_volume[],\n        const long &volume[],\n        const int &spread[])

{
    //--- calculate the indicator according to the flag value
    if(InpPrices)
        CandleSizeOnBuffer(rates_total,prev_calculated,open,close,ExtBuffer);
    else
        CandleSizeOnBuffer(rates_total,prev_calculated,high,low,ExtBuffer);
    //--- return value of prev_calculated for next call
    return(rates_total);
}
```

See also

[Access to timeseries](#), [ArraySetAsSeries](#)

ArrayInitialize

The function initializes a numeric array by a preset value.

For initialization of an array of char type

```
int ArrayInitialize(  
    char    array[],    // initialized array  
    char    value       // value that will be set  
);
```

For initialization of an array of short type

```
int ArrayInitialize(  
    short   array[],    // initialized array  
    short   value       // value that will be set  
);
```

For initialization of an array of int type

```
int ArrayInitialize(  
    int     array[],    // initialized array  
    int     value       // value that will be set  
);
```

For initialization of an array of long type

```
int ArrayInitialize(  
    long    array[],    // initialized array  
    long    value       // value that will be set  
);
```

For initialization of an array of float type

```
int ArrayInitialize(  
    float   array[],    // initialized array  
    float   value       // value that will be set  
);
```

For initialization of an array of double type

```
int ArrayInitialize(  
    double  array[],    // initialized array  
    double  value       // value that will be set  
);
```

For initialization of an array of bool type

```
int ArrayInitialize(  
    bool    array[],    // initialized array  
    bool    value       // value that will be set  
);
```


For initialization of an array of uint type

```
int ArrayInitialize(
    uint    array[],    // initialized array
    uint    value       // value that will be set
);
```

Parameters

array[]

[out] Numeric array that should be initialized.

value

[in] New value that should be set to all array elements.

Return Value

Number of initialized elements.

Note

The [ArrayResize\(\)](#) function allows to set size of an array with a reserve for further expansion without the physical relocation of memory. It is implemented for the better performance, because the operations of memory relocation are reasonably slow.

Initialization of the array using [ArrayInitialize\(array, init_val\)](#) doesn't mean the initialization with the same value of reserve elements allocated for this array. At further expanding of the *array* using the [ArrayResize\(\)](#) function, the elements will be added at the end of the array, their values will be undefined and in most cases will not be equal to *init_value*.

Example:

```
void OnStart()
{
    //--- dynamic array
    double array[];
    //--- let's set the array size for 100 elements and reserve a buffer for another 10 e
    ArrayResize(array,100,10);
    //--- initialize the array elements with EMPTY_VALUE=DBL_MAX value
    ArrayInitialize(array,EMPTY_VALUE);
    Print("Values of 10 last elements after initialization");
    for(int i=90;i<100;i++) printf("array[%d] = %G",i,array[i]);
    //--- expand the array by 5 elements
    ArrayResize(array,105);
    Print("Values of 10 last elements after ArrayResize(array,105)");
    //--- values of 5 last elements are obtained from reserve buffer
    for(int i=95;i<105;i++) printf("array[%d] = %G",i,array[i]);
}
```


ArrayFill

The function fills an array with the specified value.

```
void ArrayFill(
    void& array[],      // array
    int start,          // starting index
    int count,          // number of elements to fill
    void value          // value
);
```

Parameters

array[]

[out] Array of simple type ([char](#), [uchar](#), [short](#), [ushort](#), [int](#), [uint](#), [long](#), [ulong](#), [bool](#), [color](#), [datetime](#), [float](#), [double](#)).

start

[in] Starting index. In such a case, specified [AS_SERIES flag](#) is ignored.

count

[in] Number of elements to fill.

value

[in] Value to fill the array with.

Return Value

No return value.

Note

When ArrayFill() function is called, normal indexation direction (from left to right) is always implied. It means that the change of the order of access to the array elements using [ArraySetAsSeries\(\)](#) function is ignored.

A multidimensional array is shown as one-dimensional when processed by ArrayFill() function. For example, array[2][4] is processed as array[8]. Therefore, you may specify the initial element's index to be equal to 5 when working with this array. Thus, the call of ArrayFill(array, 5, 2, 3.14) for array[2][4] fills array[1][1] and array[1][2] elements with 3.14.

Example:

```
void OnStart()
{
    //--- declare dynamic array
    int a[];
    //--- set size
    ArrayResize(a,10);
    //--- fill first 5 elements with 123
    ArrayFill(a,0,5,123);
    //--- fill next 5 elements with 456
    ArrayFill(a,5,5,456);
}
```



```
//--- show values
for(int i=0;i<ArraySize(a);i++) printf("a[%d] = %d",i,a[i]);
}
```


ArrayIsDynamic

The function checks whether an array is dynamic.

```
bool ArrayIsDynamic(
    const void& array[]    // checked array
);
```

Parameters

`array[]`
 [in] Checked array.

Return Value

It returns true if the selected array is [dynamic](#), otherwise it returns false.

Example:

```
#property description "This indicator does not calculate values. It makes a single att
#property description "apply the call of ArrayFree() function to three arrays: dynamic
#property description "an indicator buffer. Results are shown in Experts journal."
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- global variables
double ExtDynamic[]; // dynamic array
double ExtStatic[100]; // static array
bool ExtFlag=true; // flag
double ExtBuff[]; // indicator buffer
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- allocate memory for the array
    ArrayResize(ExtDynamic,100);
    //--- indicator buffers mapping
    SetIndexBuffer(0,ExtBuff);
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const int begin,
               const double &price[])
```



```

{
//--- perform a single analysis
if(ExtFlag)
{
    //--- attempt to free memory for arrays
    //--- 1. Dynamic array
    Print("+=====+");
    Print("1. Check dynamic array:");
    Print("Size before memory is freed = ",ArraySize(ExtDynamic));
    Print("Is this a dynamic array = ",ArrayIsDynamic(ExtDynamic) ? "Yes" : "No");
    //--- attempt to free array memory
    ArrayFree(ExtDynamic);
    Print("Size after memory is freed = ",ArraySize(ExtDynamic));
    //--- 2. Static array
    Print("2. Check static array:");
    Print("Size before memory is freed = ",ArraySize(ExtStatic));
    Print("Is this a dynamic array = ",ArrayIsDynamic(ExtStatic) ? "Yes" : "No");
    //--- attempt to free array memory
    ArrayFree(ExtStatic);
    Print("Size after memory is freed = ",ArraySize(ExtStatic));
    //--- 3. Indicator buffer
    Print("3. Check indicator buffer:");
    Print("Size before memory is freed = ",ArraySize(ExtBuff));
    Print("Is this a dynamic array = ",ArrayIsDynamic(ExtBuff) ? "Yes" : "No");
    //--- attempt to free array memory
    ArrayFree(ExtBuff);
    Print("Size after memory is freed = ",ArraySize(ExtBuff));
    //--- change the flag value
    ExtFlag=false;
}
//--- return value of prev_calculated for next call
return(rates_total);
}

```

See also

[Access to timeseries and indicators](#)

ArrayIsSeries

The function checks whether an array is a timeseries.

```
bool ArrayIsSeries(
    const void& array[]    // checked array
);
```

Parameters

`array[]`
 [in] Checked array.

Return Value

It returns true, if a checked array is an array timeseries, otherwise it returns false. Arrays passed as a parameter to the [OnCalculate\(\)](#) function must be checked for the order of accessing the array elements by [ArrayGetAsSeries\(\)](#).

Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double Label1Buffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
    //---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```



```
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//---
        if(ArrayIsSeries(open))
            Print("open[] is timeseries");
        else
            Print("open[] is not timeseries!!!");
//--- return value of prev_calculated for next call
        return(rates_total);
    }
```

See also

[Access to timeseries and indicators](#)

ArrayMaximum

Searches for the largest element in the first dimension of a multidimensional numeric array.

```
int ArrayMaximum(
    const void& array[],           // array for search
    int start=0,                  // index to start checking with
    int count=WHOLE_ARRAY         // number of checked elements
);
```

Parameters

array[]

[in] A numeric array, in which search is made.

start=0

[in] Index to start checking with.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=[WHOLE_ARRAY](#)).

Return Value

The function returns an index of a found element taking into account the array [serial](#). In case of failure it returns -1.

Note

The [AS_SERIES](#) flag value is taken into account while searching for a maximum.

Functions ArrayMaximum and ArrayMinimum accept any-dimensional arrays as a parameter. However, searching is always applied to the first (zero) dimension.

Example:

```
#property description "The indicator displays larger timeframe's candlesticks on the c
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots 8
//---- plot 1
#property indicator_label1 "BearBody"
#property indicator_color1 clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2 "BearBodyEnd"
#property indicator_color2 clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3 "BearShadow"
#property indicator_color3 clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4 "BearShadowEnd"
#property indicator_color4 clrSalmon,clrSalmon
```



```

//---- plot 5
#property indicator_label5 "BullBody"
#property indicator_color5 clrOlive,clrOlive
//---- plot 6
#property indicator_label6 "BullBodyEnd"
#property indicator_color6 clrOlive,clrOlive
//---- plot 7
#property indicator_label7 "BullShadow"
#property indicator_color7 clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8 "BullShadowEnd"
#property indicator_color8 clrSkyBlue,clrSkyBlue
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4; // Time frame for the indicator
input datetime InpDateStart=D'2013.01.01 00:00'; // Analysis start date
//--- indicator buffers for bearish candlesticks
double ExtBearBodyFirst[];
double ExtBearBodySecond[];
double ExtBearBodyEndFirst[];
double ExtBearBodyEndSecond[];
double ExtBearShadowFirst[];
double ExtBearShadowSecond[];
double ExtBearShadowEndFirst[];
double ExtBearShadowEndSecond[];
//--- indicator buffers for bullish candlesticks
double ExtBullBodyFirst[];
double ExtBullBodySecond[];
double ExtBullBodyEndFirst[];
double ExtBullBodyEndSecond[];
double ExtBullShadowFirst[];
double ExtBullShadowSecond[];
double ExtBullShadowEndFirst[];
double ExtBullShadowEndSecond[];
//--- global variables
datetime ExtTimeBuff[]; // larger time frame's time buffer
int ExtSize=0; // time buffer size
int ExtCount=0; // index inside time buffer
int ExtStartPos=0; // initial position for the indicator calculation
bool ExtStartFlag=true; // auxiliary flag for receiving the initial position
datetime ExtCurrentTime[1]; // last time of the larger time frame's bar generation
datetime ExtLastTime; // last time from the larger time frame, for which the candlestick is generated
bool ExtBearFlag=true; // flag for defining the order of writing the data to bearish candlesticks
bool ExtBullFlag=true; // flag for defining the order of writing the data to bullish candlesticks
int ExtIndexMax=0; // index of the maximum element in the array
int ExtIndexMin=0; // index of the minimum element in the array
int ExtDirectionFlag=0; // price movement direction for the current candlestick
//--- shift between the candlestick's open and close price for correct drawing

```



```

const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+-----+
//| Filling the basic part of the candlestick |
//+-----+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
{
    //--- find the index of the maximum and minimum elements in the arrays
    index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // maximum in High
    index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // minimum in Low
    //--- define how many bars from the current time frame are to be filled out
    int count=fill_index-start+1;
    //--- if the close price at the first bar exceeds the one at the last bar, the candles
    if(open[start]>close[last])
    {
        //--- if the candlestick has been bullish before that, clear the values of bullish
        if(ExtDirectionFlag!=-1)
            ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
        //--- bearish candlestick
        ExtDirectionFlag=-1;
        //--- generate the candlestick
        FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                       close[last],high[index_max],low[index_min],start,count,ExtBearFlag);
        //--- exit the function
        return;
    }
    //--- if the close price at the first bar is less than the one at the last bar, the candles
    if(open[start]<close[last])
    {
        //--- if the candlestick has been bearish before that, clear the values of bearish
        if(ExtDirectionFlag!=1)
            ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond);
        //--- bullish candlestick
        ExtDirectionFlag=1;
        //--- generate the candlestick
        FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,
                       open[start],high[index_max],low[index_min],start,count,ExtBullFlag);
        //--- exit the function
        return;
    }
    //--- if you are in this part of the function, the open price at the first bar is equal
    //--- the close price at the last bar; such candlestick is considered bearish
    //--- if the candlestick has been bullish before that, clear the values of bullish indicators
    if(ExtDirectionFlag!=-1)
        ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
    //--- bearish candlestick
    ExtDirectionFlag=-1;

```



```

//--- if close and open prices are equal, use the shift for correct display
if(high[index_max]!=low[index_min])
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
        open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start,count,ExtBearFlag);
else
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
        open[start],open[start]-ExtEmptyBodySize,high[index_max],
        high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
}
//+-----+
//| Fill out the end of the candlestick |
//+-----+
void FillCandleEnd(const double &open[],const double &close[],
    const double &high[],const double &low[],
    const int start,const int last,const int fill_index,
    const int index_max,const int index_min)
{
//--- do not draw in case of a single bar
if(last-start==0)
    return;
//--- if the close price at the first bar exceeds the one at the last bar, the candlestick is bearish
if(open[start]>close[last])
{
    //--- generate the end of the candlestick
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
        open[start],close[last],high[index_max],low[index_min],fill_index,
        start,count,ExtBearFlag);
    //--- exit the function
    return;
}
//--- if the close price at the first bar is less than the one at the last bar, the candlestick is bullish
if(open[start]<close[last])
{
    //--- generate the end of the candlestick
    FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,ExtBullShadowEndSecond,
        close[last],open[start],high[index_max],low[index_min],fill_index,
        start,count,ExtBullFlag);
    //--- exit the function
    return;
}
//--- if you are in this part of the function, the open price at the first bar is equal to the close price at the last bar; such candlestick is considered bearish
//--- generate the end of the candlestick
if(high[index_max]!=low[index_min])
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
        open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_index,
        start,count,ExtBearFlag);
else
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
        open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEmptyBodySize,fill_index,
        start,count,ExtBearFlag);
}
//+-----+

```



```

//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- check the indicator period
    if(!CheckPeriod((int)Period(),(int)InpPeriod))
        return(INIT_PARAMETERS_INCORRECT);
//--- display price data in the foreground
    ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- binding indicator buffers
    SetIndexBuffer(0,ExtBearBodyFirst);
    SetIndexBuffer(1,ExtBearBodySecond);
    SetIndexBuffer(2,ExtBearBodyEndFirst);
    SetIndexBuffer(3,ExtBearBodyEndSecond);
    SetIndexBuffer(4,ExtBearShadowFirst);
    SetIndexBuffer(5,ExtBearShadowSecond);
    SetIndexBuffer(6,ExtBearShadowEndFirst);
    SetIndexBuffer(7,ExtBearShadowEndSecond);
    SetIndexBuffer(8,ExtBullBodyFirst);
    SetIndexBuffer(9,ExtBullBodySecond);
    SetIndexBuffer(10,ExtBullBodyEndFirst);
    SetIndexBuffer(11,ExtBullBodyEndSecond);
    SetIndexBuffer(12,ExtBullShadowFirst);
    SetIndexBuffer(13,ExtBullShadowSecond);
    SetIndexBuffer(14,ExtBullShadowEndFirst);
    SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- set some property values for creating the indicator
    for(int i=0;i<8;i++)
    {
        PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // graphical construction type
        PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // drawing line style
        PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // drawing line width
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- in case there are no calculated bars yet
if(prev_calculated==0)
{
    //--- receive larger time frame's bars arrival time
    if(!GetTimeData())
        return(0);
}
//--- set direct indexing
ArraySetAsSeries(time,false);
ArraySetAsSeries(high,false);
ArraySetAsSeries(low,false);
ArraySetAsSeries(open,false);
ArraySetAsSeries(close,false);
//--- start variable for calculation of bars
int start=prev_calculated;
//--- if the bar is generated, recalculate the indicator value on it
if(start!=0 && start==rates_total)
    start--;
//--- the loop for calculating the indicator values
for(int i=start;i<rates_total;i++)
{
    //--- fill i elements of the indicator buffers by empty values
    FillIndicatorBuffers(i);
    //--- perform calculation for bars starting from InpDateStart date
    if(time[i]>=InpDateStart)
    {
        //--- define position, from which the values are to be displayed, for the first bar
        if(ExtStartFlag)
        {
            //--- store the number of the initial bar
            ExtStartPos=i;
            //--- define the first date from the larger time frame exceeding time[i]
            while(time[i]>=ExtTimeBuff[ExtCount])
                if(ExtCount<ExtSize-1)
                    ExtCount++;
            //--- change the value of the flag in order not to run into this block again
            ExtStartFlag=false;
        }
        //--- check if there are still any elements in the array
        if(ExtCount<ExtSize)
        {
            //--- wait for the current time frame's value to reach the larger time frame
            if(time[i]>=ExtTimeBuff[ExtCount])
            {
                //--- draw the main part of the candlestick (without filling out the area)
                FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,ExtIndexMin);
                //--- fill out the end of the candlestick (the area between the last array element and the start of the bar)
                FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
            }
        }
    }
}
}

```



```

        //--- shift the initial position for drawing the next candlestick
        ExtStartPos=i;
        //--- increase the array counter
        ExtCount++;
    }
    else
        continue;
}
else
{
    //--- reset the array values
    ResetLastError();
    //--- receive the last date from the larger time frame
    if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
    {
        Print("Data copy error, code = ",GetLastError());
        return(0);
    }
    //--- if the new date is later, stop generating the candlestick
    if(ExtCurrentTime[0]>ExtLastTime)
    {
        //--- clear the area between the last and penultimate bars in the main
        ClearEndOfBodyMain(i-1);
        //--- fill out the area using auxiliary indicator buffers
        FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
        //--- shift the initial position for drawing the next candlestick
        ExtStartPos=i;
        //--- reset price direction flag
        ExtDirectionFlag=0;
        //--- store the new last date
        ExtLastTime=ExtCurrentTime[0];
    }
    else
    {
        //--- generate the candlestick
        FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndexMin);
    }
}
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Check correctness of the specified indicator period |
//+-----+
bool CheckPeriod(int current_period,int high_period)
{
    //--- the indicator period should exceed the timeframe on which it is displayed

```



```

    if(current_period>=high_period)
    {
        Print("Error! The value of the indicator period should exceed the value of the c
        return(false);
    }
//--- if the indicator period is one week or month, the period is correct
    if(high_period>32768)
        return(true);
//--- convert period values to minutes
    if(high_period>30)
        high_period=(high_period-16384)*60;
    if(current_period>30)
        current_period=(current_period-16384)*60;
//--- the indicator period should be multiple of the time frame it is displayed on
    if(high_period%current_period!=0)
    {
        Print("Error! The value of the indicator period should be multiple of the value
        return(false);
    }
//--- the indicator period should exceed the time frame it is displayed on 3 or more t
    if(high_period/current_period<3)
    {
        Print("Error! The indicator period should exceed the current time frame 3 or mor
        return(false);
    }
//--- the indicator period is correct for the current time frame
    return(true);
}
//+-----+
//| Receive time data from the larger time frame |
//+-----+
bool GetTimeData(void)
{
//--- reset the error value
    ResetLastError();
//--- copy all data for the current time
    if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
    {
        //--- receive the error code
        int code=GetLastError();
        //--- print out the error message
        PrintFormat("Data copy error! %s",code==4401
            ? "History is still being uploaded!"
            : "Code = "+IntegerToString(code));
        //--- return false to make a repeated attempt to download data
        return(false);
    }
//--- receive the array size
    ExtSize=ArraySize(ExtTimeBuff);

```



```

//--- set the loop index for the array to zero
ExtCount=0;
//--- set the current candlestick's position on the time frame to zero
ExtStartPos=0;
ExtStartFlag=true;
//--- store the last time value from the larger time frame
ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- successful execution
return(true);
}
//+-----+
//| Function forms the main part of the candlestick. Depending on the flag's |
//| value, the function defines what data and arrays are                    |
//| to be used for correct display.                                         |
//+-----+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
{
//--- check the flag's value
if(flag)
{
//--- generate the candlestick's body
FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
//--- generate the candlestick's shadow
FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
}
else
{
//--- generate the candlestick's body
FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
//--- generate the candlestick's shadow
FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
}
}
//+-----+
//| The function forms the end of the candlestick. Depending on the flag's value, |
//| the function defines what data and arrays are                                |
//| to be used for correct display.                                             |
//+-----+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
                   const int end,bool &flag)
{
//--- check the flag's value

```



```

    if(flag)
    {
        //--- generate the end of the candlestick's body
        FormEnd(body_fst,body_snd,fst_value,snd_value,end);
        //--- generate the end of the candlestick's shadow
        FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
        //--- change the flag's value to the opposite one
        flag=false;
    }
    else
    {
        //--- generate the end of the candlestick's body
        FormEnd(body_fst,body_snd,snd_value,fst_value,end);
        //--- generate the end of the candlestick's shadow
        FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
        //--- change the flag's value to the opposite one
        flag=true;
    }
}

//+-----+
//| Clear the end of the candlestick (the area between the last and the penultimate |
//| bar)                                     |
//+-----+
void ClearEndOfBodyMain(const int ind)
{
    ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
    ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
}

//+-----+
//| Clear the candlestick                                     |
//+-----+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
{
    //--- check
    if(count!=0)
    {
        //--- fill indicator buffers with empty values
        ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
    }
}

//+-----+
//| Generate the main part of the candlestick                                     |
//+-----+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)

```



```

{
//--- check
if(count!=0)
{
    //--- fill indicator buffers with values
    ArrayFill(fst,start,count,fst_value);
    ArrayFill(snd,start,count,snd_value);
}
}

//+-----+
//| Generate the end of the candlestick |
//+-----+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
{
//--- fill indicator buffers with values
    ArrayFill(fst,last-1,2,fst_value);
    ArrayFill(snd,last-1,2,snd_value);
}

//+-----+
//| Fill i element of the indicator buffers by empty values |
//+-----+
void FillIndicatorBuffers(const int i)
{
//--- set an empty value in the cell of the indicator buffers
    ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
}

```


ArrayMinimum

Searches for the lowest element in the first dimension of a multidimensional numeric array.

```
int ArrayMinimum(
    const void& array[],           // array for search
    int start=0,                  // index to start checking with
    int count=WHOLE_ARRAY         // number of checked elements
);
```

Parameters

array[]

[in] A numeric array, in which search is made.

start=0

[in] Index to start checking with.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=[WHOLE_ARRAY](#)).

Return Value

The function returns an index of a found element taking into account the array [serial](#). In case of failure it returns -1.

Note

The [AS_SERIES](#) flag value is taken into account while searching for a minimum.

Functions `ArrayMaximum` and `ArrayMinimum` accept any-dimensional arrays as a parameter. However, searching is always applied to the first (zero) dimension.

Example:

```
#property description "The indicator displays larger timeframe's candlesticks on the c
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots 8
//---- plot 1
#property indicator_label1 "BearBody"
#property indicator_color1 clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2 "BearBodyEnd"
#property indicator_color2 clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3 "BearShadow"
#property indicator_color3 clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4 "BearShadowEnd"
#property indicator_color4 clrSalmon,clrSalmon
```



```

//---- plot 5
#property indicator_label5 "BullBody"
#property indicator_color5 clrOlive,clrOlive
//---- plot 6
#property indicator_label6 "BullBodyEnd"
#property indicator_color6 clrOlive,clrOlive
//---- plot 7
#property indicator_label7 "BullShadow"
#property indicator_color7 clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8 "BullShadowEnd"
#property indicator_color8 clrSkyBlue,clrSkyBlue
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4; // Time frame for the indicator
input datetime InpDateStart=D'2013.01.01 00:00'; // Analysis start date
//--- indicator buffers for bearish candlesticks
double ExtBearBodyFirst[];
double ExtBearBodySecond[];
double ExtBearBodyEndFirst[];
double ExtBearBodyEndSecond[];
double ExtBearShadowFirst[];
double ExtBearShadowSecond[];
double ExtBearShadowEndFirst[];
double ExtBearShadowEndSecond[];
//--- indicator buffers for bullish candlesticks
double ExtBullBodyFirst[];
double ExtBullBodySecond[];
double ExtBullBodyEndFirst[];
double ExtBullBodyEndSecond[];
double ExtBullShadowFirst[];
double ExtBullShadowSecond[];
double ExtBullShadowEndFirst[];
double ExtBullShadowEndSecond[];
//--- global variables
datetime ExtTimeBuff[]; // larger time frame's time buffer
int ExtSize=0; // time buffer size
int ExtCount=0; // index inside time buffer
int ExtStartPos=0; // initial position for the indicator calculation
bool ExtStartFlag=true; // auxiliary flag for receiving the initial position
datetime ExtCurrentTime[1]; // last time of the larger time frame's bar generation
datetime ExtLastTime; // last time from the larger time frame, for which the candlestick is generated
bool ExtBearFlag=true; // flag for defining the order of writing the data to bearish candlesticks
bool ExtBullFlag=true; // flag for defining the order of writing the data to bullish candlesticks
int ExtIndexMax=0; // index of the maximum element in the array
int ExtIndexMin=0; // index of the minimum element in the array
int ExtDirectionFlag=0; // price movement direction for the current candlestick
//--- shift between the candlestick's open and close price for correct drawing

```



```

const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+-----+
//| Filling the basic part of the candlestick |
//+-----+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
{
    //--- find the index of the maximum and minimum elements in the arrays
    index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // maximum in High
    index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // minimum in Low
    //--- define how many bars from the current time frame are to be filled out
    int count=fill_index-start+1;
    //--- if the close price at the first bar exceeds the one at the last bar, the candles
    if(open[start]>close[last])
    {
        //--- if the candlestick has been bullish before that, clear the values of bullish
        if(ExtDirectionFlag!=-1)
            ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
        //--- bearish candlestick
        ExtDirectionFlag=-1;
        //--- generate the candlestick
        FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                       close[last],high[index_max],low[index_min],start,count,ExtBearFlag);
        //--- exit the function
        return;
    }
    //--- if the close price at the first bar is less than the one at the last bar, the candles
    if(open[start]<close[last])
    {
        //--- if the candlestick has been bearish before that, clear the values of bearish
        if(ExtDirectionFlag!=1)
            ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond);
        //--- bullish candlestick
        ExtDirectionFlag=1;
        //--- generate the candlestick
        FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,
                       open[start],high[index_max],low[index_min],start,count,ExtBullFlag);
        //--- exit the function
        return;
    }
    //--- if you are in this part of the function, the open price at the first bar is equal to
    //--- the close price at the last bar; such candlestick is considered bearish
    //--- if the candlestick has been bullish before that, clear the values of bullish indicators
    if(ExtDirectionFlag!=-1)
        ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
    //--- bearish candlestick
    ExtDirectionFlag=-1;

```



```

//--- if close and open prices are equal, use the shift for correct display
if (high[index_max] != low[index_min])
    FormCandleMain (ExtBearBodyFirst, ExtBearBodySecond, ExtBearShadowFirst, ExtBearShadowSecond,
        open[start] - ExtEmptyBodySize, high[index_max], low[index_min], start, count, ExtBearFlag);
else
    FormCandleMain (ExtBearBodyFirst, ExtBearBodySecond, ExtBearShadowFirst, ExtBearShadowSecond,
        open[start], open[start] - ExtEmptyBodySize, high[index_max],
        high[index_max] - ExtEmptyBodySize, start, count, ExtBearFlag);
}
//+-----+
//| Fill out the end of the candlestick |
//+-----+
void FillCandleEnd(const double &open[], const double &close[],
    const double &high[], const double &low[],
    const int start, const int last, const int fill_index,
    const int index_max, const int index_min)
{
//--- do not draw in case of a single bar
if (last - start == 0)
    return;
//--- if the close price at the first bar exceeds the one at the last bar, the candlestick is bearish
if (open[start] > close[last])
{
    //--- generate the end of the candlestick
    FormCandleEnd (ExtBearBodyEndFirst, ExtBearBodyEndSecond, ExtBearShadowEndFirst, ExtBearShadowEndSecond,
        open[start], close[last], high[index_max], low[index_min], fill_index,
        start, count, ExtBearFlag);
    //--- exit the function
    return;
}
//--- if the close price at the first bar is less than the one at the last bar, the candlestick is bullish
if (open[start] < close[last])
{
    //--- generate the end of the candlestick
    FormCandleEnd (ExtBullBodyEndFirst, ExtBullBodyEndSecond, ExtBullShadowEndFirst, ExtBullShadowEndSecond,
        close[last], open[start], high[index_max], low[index_min], fill_index,
        start, count, ExtBullFlag);
    //--- exit the function
    return;
}
//--- if you are in this part of the function, the open price at the first bar is equal to
//--- the close price at the last bar; such candlestick is considered bearish
//--- generate the end of the candlestick
if (high[index_max] != low[index_min])
    FormCandleEnd (ExtBearBodyEndFirst, ExtBearBodyEndSecond, ExtBearShadowEndFirst, ExtBearShadowEndSecond,
        open[start] - ExtEmptyBodySize, high[index_max], low[index_min], fill_index,
        start, count, ExtBearFlag);
else
    FormCandleEnd (ExtBearBodyEndFirst, ExtBearBodyEndSecond, ExtBearShadowEndFirst, ExtBearShadowEndSecond,
        open[start] - ExtEmptyBodySize, high[index_max], high[index_max] - ExtEmptyBodySize, fill_index,
        start, count, ExtBearFlag);
}
//+-----+

```



```

//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- check the indicator period
    if(!CheckPeriod((int)Period(),(int)InpPeriod))
        return(INIT_PARAMETERS_INCORRECT);
//--- display price data in the foreground
    ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- binding indicator buffers
    SetIndexBuffer(0,ExtBearBodyFirst);
    SetIndexBuffer(1,ExtBearBodySecond);
    SetIndexBuffer(2,ExtBearBodyEndFirst);
    SetIndexBuffer(3,ExtBearBodyEndSecond);
    SetIndexBuffer(4,ExtBearShadowFirst);
    SetIndexBuffer(5,ExtBearShadowSecond);
    SetIndexBuffer(6,ExtBearShadowEndFirst);
    SetIndexBuffer(7,ExtBearShadowEndSecond);
    SetIndexBuffer(8,ExtBullBodyFirst);
    SetIndexBuffer(9,ExtBullBodySecond);
    SetIndexBuffer(10,ExtBullBodyEndFirst);
    SetIndexBuffer(11,ExtBullBodyEndSecond);
    SetIndexBuffer(12,ExtBullShadowFirst);
    SetIndexBuffer(13,ExtBullShadowSecond);
    SetIndexBuffer(14,ExtBullShadowEndFirst);
    SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- set some property values for creating the indicator
    for(int i=0;i<8;i++)
    {
        PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // graphical construction type
        PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // drawing line style
        PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // drawing line width
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- in case there are no calculated bars yet
if(prev_calculated==0)
{
    //--- receive larger time frame's bars arrival time
    if(!GetTimeData())
        return(0);
}
//--- set direct indexing
ArraySetAsSeries(time,false);
ArraySetAsSeries(high,false);
ArraySetAsSeries(low,false);
ArraySetAsSeries(open,false);
ArraySetAsSeries(close,false);
//--- start variable for calculation of bars
int start=prev_calculated;
//--- if the bar is generated, recalculate the indicator value on it
if(start!=0 && start==rates_total)
    start--;
//--- the loop for calculating the indicator values
for(int i=start;i<rates_total;i++)
{
    //--- fill i elements of the indicator buffers by empty values
    FillIndicatorBuffers(i);
    //--- perform calculation for bars starting from InpDateStart date
    if(time[i]>=InpDateStart)
    {
        //--- define position, from which the values are to be displayed, for the first bar
        if(ExtStartFlag)
        {
            //--- store the number of the initial bar
            ExtStartPos=i;
            //--- define the first date from the larger time frame exceeding time[i]
            while(time[i]>=ExtTimeBuff[ExtCount])
                if(ExtCount<ExtSize-1)
                    ExtCount++;
            //--- change the value of the flag in order not to run into this block again
            ExtStartFlag=false;
        }
        //--- check if there are still any elements in the array
        if(ExtCount<ExtSize)
        {
            //--- wait for the current time frame's value to reach the larger time frame
            if(time[i]>=ExtTimeBuff[ExtCount])
            {
                //--- draw the main part of the candlestick (without filling out the area)
                FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,ExtIndexMin);
                //--- fill out the end of the candlestick (the area between the last array element and the start of the candlestick)
                FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
            }
        }
    }
}
}

```



```

        //--- shift the initial position for drawing the next candlestick
        ExtStartPos=i;
        //--- increase the array counter
        ExtCount++;
    }
    else
        continue;
}
else
{
    //--- reset the array values
    ResetLastError();
    //--- receive the last date from the larger time frame
    if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
    {
        Print("Data copy error, code = ",GetLastError());
        return(0);
    }
    //--- if the new date is later, stop generating the candlestick
    if(ExtCurrentTime[0]>ExtLastTime)
    {
        //--- clear the area between the last and penultimate bars in the main
        ClearEndOfBodyMain(i-1);
        //--- fill out the area using auxiliary indicator buffers
        FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
        //--- shift the initial position for drawing the next candlestick
        ExtStartPos=i;
        //--- reset price direction flag
        ExtDirectionFlag=0;
        //--- store the new last date
        ExtLastTime=ExtCurrentTime[0];
    }
    else
    {
        //--- generate the candlestick
        FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndexMin);
    }
}
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Check correctness of the specified indicator period |
//+-----+
bool CheckPeriod(int current_period,int high_period)
{
    //--- the indicator period should exceed the timeframe on which it is displayed

```



```

    if(current_period>=high_period)
    {
        Print("Error! The value of the indicator period should exceed the value of the c
        return(false);
    }
//--- if the indicator period is one week or month, the period is correct
    if(high_period>32768)
        return(true);
//--- convert period values to minutes
    if(high_period>30)
        high_period=(high_period-16384)*60;
    if(current_period>30)
        current_period=(current_period-16384)*60;
//--- the indicator period should be multiple of the time frame it is displayed on
    if(high_period%current_period!=0)
    {
        Print("Error! The value of the indicator period should be multiple of the value
        return(false);
    }
//--- the indicator period should exceed the time frame it is displayed on 3 or more t
    if(high_period/current_period<3)
    {
        Print("Error! The indicator period should exceed the current time frame 3 or mor
        return(false);
    }
//--- the indicator period is correct for the current time frame
    return(true);
}
//+-----+
//| Receive time data from the larger time frame |
//+-----+
bool GetTimeData(void)
{
//--- reset the error value
    ResetLastError();
//--- copy all data for the current time
    if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
    {
        //--- receive the error code
        int code=GetLastError();
        //--- print out the error message
        PrintFormat("Data copy error! %s",code==4401
            ? "History is still being uploaded!"
            : "Code = "+IntegerToString(code));
        //--- return false to make a repeated attempt to download data
        return(false);
    }
//--- receive the array size
    ExtSize=ArraySize(ExtTimeBuff);

```



```

//--- set the loop index for the array to zero
ExtCount=0;
//--- set the current candlestick's position on the time frame to zero
ExtStartPos=0;
ExtStartFlag=true;
//--- store the last time value from the larger time frame
ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- successful execution
return(true);
}
//+-----+
//| Function forms the main part of the candlestick. Depending on the flag's |
//| value, the function defines what data and arrays are |
//| to be used for correct display. |
//+-----+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
{
//--- check the flag's value
if(flag)
{
//--- generate the candlestick's body
FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
//--- generate the candlestick's shadow
FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
}
else
{
//--- generate the candlestick's body
FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
//--- generate the candlestick's shadow
FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
}
}
//+-----+
//| The function forms the end of the candlestick. Depending on the flag's value, |
//| the function defines what data and arrays are |
//| to be used for correct display. |
//+-----+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                  double &shadow_fst[],double &shadow_snd[],
                  const double fst_value,const double snd_value,
                  const double fst_extremum,const double snd_extremum,
                  const int end,bool &flag)
{
//--- check the flag's value

```



```

    if(flag)
    {
        //--- generate the end of the candlestick's body
        FormEnd(body_fst,body_snd,fst_value,snd_value,end);
        //--- generate the end of the candlestick's shadow
        FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
        //--- change the flag's value to the opposite one
        flag=false;
    }
    else
    {
        //--- generate the end of the candlestick's body
        FormEnd(body_fst,body_snd,snd_value,fst_value,end);
        //--- generate the end of the candlestick's shadow
        FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
        //--- change the flag's value to the opposite one
        flag=true;
    }
}

//+-----+
//| Clear the end of the candlestick (the area between the last and the penultimate
//| bar)
//+-----+

void ClearEndOfBodyMain(const int ind)
{
    ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
    ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
}

//+-----+
//| Clear the candlestick
//+-----+

void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
{
    //--- check
    if(count!=0)
    {
        //--- fill indicator buffers with empty values
        ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
    }
}

//+-----+
//| Generate the main part of the candlestick
//+-----+

void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)

```



```

{
//--- check
if(count!=0)
{
    //--- fill indicator buffers with values
    ArrayFill(fst,start,count,fst_value);
    ArrayFill(snd,start,count,snd_value);
}
}

//+-----+
//| Generate the end of the candlestick |
//+-----+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
{
//--- fill indicator buffers with values
    ArrayFill(fst,last-1,2,fst_value);
    ArrayFill(snd,last-1,2,snd_value);
}

//+-----+
//| Fill i element of the indicator buffers by empty values |
//+-----+
void FillIndicatorBuffers(const int i)
{
//--- set an empty value in the cell of the indicator buffers
    ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
}

```


ArrayPrint

Prints an array of a simple type or a simple structure into journal.

```
void ArrayPrint(
    const void&   array[],           // printed array
    uint          digits=_Digits,    // number of decimal places
    const string  separator=NULL,    // separator of the structure field values
    ulong         start=0,           // first printed element index
    ulong         count=WHOLE_ARRAY, // number of printed elements
    ulong         flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_ALIGN
);
```

Parameters

array[]

[in] Array of a simple type or a [simple structure](#).

digits=_Digits

[in] The number of decimal places for real types. The default value is [_Digits](#).

separator=NULL

[in] Separator of the structure element field values. The default value [NULL](#) means an empty line. A space is used as a separator in that case.

start=0

[in] The index of the first printed array element. It is printed from the zero index by default.

count=WHOLE_ARRAY

[in] Number of the array elements to be printed. The entire array is displayed by default (count=[WHOLE_ARRAY](#)).

flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_ALIGN

[in] Combination of flags setting the output mode. All flags are enabled by default:

- ARRAYPRINT_HEADER - print headers for the structure array
- ARRAYPRINT_INDEX - print index at the left side
- ARRAYPRINT_LIMIT - print only the first 100 and the last 100 array elements. Use if you want to print only a part of a large array.
- ARRAYPRINT_ALIGN - enable alignment of the printed values - numbers are aligned to the right, while lines to the left.
- ARRAYPRINT_DATE - when printing datetime, print the date in the dd.mm.yyyy format
- ARRAYPRINT_MINUTES - when printing datetime, print the time in the HH:MM format
- ARRAYPRINT_SECONDS - when printing datetime, print the time in the HH:MM:SS format

Return Value

No

Note

ArrayPrint() does not print all structure array fields into journal - array and [object pointer](#) fields are skipped. These columns are simply not printed for more convenient presentation. If you need to

print all structure fields, you need to write your own mass print function with the desired formatting.

Example:

```
//--- print the values of the last 10 bars
MqlRates rates[];
if(CopyRates(_Symbol,_Period,1,10,rates))
{
    ArrayPrint(rates);
    Print("Check\n[time]\t[open]\t[high]\t[low]\t[close]\t[tick_volume]\t[spread]\t[real_volume]");
    for(int i=0;i<10;i++)
    {
        PrintFormat("[%d]\t%s\t%G\t%G\t%G\t%G\t%G\t%G\t%I64d\t",i,
            TimeToString(rates[i].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS),
            rates[i].open,rates[i].high,rates[i].low,rates[i].close,
            rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
    }
}
else
    PrintFormat("CopyRates failed, error code=%d",GetLastError());
//--- example of printing
/*
            [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [real_volume]
[0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295      18110      10 17300175000
[1] 2016.11.09 05:00:00 1.12296 1.12825 1.11930 1.12747      17829       9 15632176000
[2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744      13458      10 9593492000
[3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194      15362       9 12352245000
[4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172      16833       9 12961333000
[5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052      15933       8 10720384000
[6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528      11888       9 8084811000
[7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915       7284      10 5087113000
[8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904       8710       9 6769629000
[9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263       8956       7 7192138000
Check
[time] [open] [high] [low] [close] [tick_volume] [spread] [real_volume]
[0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295 18110 10 17300175000
[1] 2016.11.09 05:00:00 1.12296 1.12825 1.1193 1.12747 17829 9 15632176000
[2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744 13458 10 9593492000
[3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194 15362 9 12352245000
[4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172 16833 9 12961333000
[5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052 15933 8 10720384000
[6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528 11888 9 8084811000
[7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915 7284 10 5087113000
[8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904 8710 9 6769629000
[9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263 8956 7 7192138000
*/
```

See also

[FileSave](#), [FileLoad](#)

ArrayRange

The function returns the number of elements in a selected array dimension.

```
int ArrayRange(  
    const void& array[], // array for check  
    int rank_index // index of dimension  
);
```

Parameters

array[]

[in] Checked array.

rank_index

[in] Index of dimension.

Return Value

Number of elements in a selected array dimension.

Note

Since indexes start at zero, the number of the array dimensions is one greater than the index of the last dimension.

Example:

```
void OnStart()  
{  
    //--- create four-dimensional array  
    double array[][5][2][4];  
    //--- set the size of the zero dimension  
    ArrayResize(array,10,10);  
    //--- print dimensions  
    int temp;  
    for(int i=0;i<4;i++)  
    {  
        //--- receive the size of i dimension  
        temp=ArrayRange(array,i);  
        //--- print  
        PrintFormat("dim = %d, range = %d",i,temp);  
    }  
    //--- Result  
    // dim = 0, range = 10  
    // dim = 1, range = 5  
    // dim = 2, range = 2  
    // dim = 3, range = 4  
}
```


ArrayResize

The function sets a new size for the first dimension

```
int ArrayResize(
    void& array[],           // array passed by reference
    int new_size,           // new array size
    int reserve_size=0      // reserve size value (excess)
);
```

Parameters

array[]

[out] Array for changing sizes.

new_size

[in] New size for the first dimension.

reserve_size=0

[in] Distributed size to get reserve.

Return Value

If executed successfully, it returns count of all elements contained in the array after resizing, otherwise, returns -1, and array is not resized.

Note

The function can be applied only to [dynamic arrays](#). It should be noted that you cannot change the size of dynamic arrays assigned as indicator buffers by the [SetIndexBuffer\(\)](#) function. For indicator buffers, all operations of resizing are performed by the runtime subsystem of the terminal.

Total amount of elements in the array cannot exceed 2147483647.

With the frequent memory allocation, it is recommended to use a third parameter that sets a reserve to reduce the number of physical memory allocations. All the subsequent calls of [ArrayResize](#) do not lead to physical reallocation of memory, but only change the size of the first array dimension within the reserved memory. It should be remembered that the third parameter will be used only during physical memory allocation. For example:

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
    ArrayResize(arr,i,1000);
```

In this case the memory will be reallocated twice, first before entering the 2000-element loop (the array size will be set to 1000), and the second time with *i* equal to 2000. If we skip the third parameter, there will be 2000 physical reallocations of memory, which will slow down the program.

Example:

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



```

{
//--- Counters
    ulong start=GetTickCount();
    ulong now;
    int count=0;
//--- An array for demonstration of a quick version
    double arr[];
    ArrayResize(arr,100000,100000);
//--- Check how fast the variant with memory reservation works
    Print("--- Test Fast: ArrayResize(arr,100000,100000)");
    for(int i=1;i<=300000;i++)
    {
        //--- Set a new array size specifying the reserve of 100,000 elements!
        ArrayResize(arr,i,100000);
        //--- When reaching a round number, show the array size and the time spent
        if(ArraySize(arr)%100000==0)
        {
            now=GetTickCount();
            count++;
            PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(arr),(now-start));
            start=now;
        }
    }
//--- Now show, how slow the version without memory reservation is
    double slow[];
    ArrayResize(slow,100000,100000);
//---
    count=0;
    start=GetTickCount();
    Print("---- Test Slow: ArrayResize(slow,100000)");
//---
    for(int i=1;i<=300000;i++)
    {
        //--- Set a new array size, but without the additional reserve
        ArrayResize(slow,i);
        //--- When reaching a round number, show the array size and the time spent
        if(ArraySize(slow)%100000==0)
        {
            now=GetTickCount();
            count++;
            PrintFormat("%d. ArraySize(slow)=%d Time=%d ms",count,ArraySize(slow),(now-start));
            start=now;
        }
    }
}

//--- A sample result of the script
/*
Test_ArrayResize (EURUSD,H1)    --- Test Fast: ArrayResize(arr,100000,100000)
Test_ArrayResize (EURUSD,H1)    1. ArraySize(arr)=100000 Time=0 ms

```



```
Test_ArrayResize (EURUSD,H1) 2. ArraySize(arr)=200000 Time=0 ms
Test_ArrayResize (EURUSD,H1) 3. ArraySize(arr)=300000 Time=0 ms
Test_ArrayResize (EURUSD,H1) ---- Test Slow: ArrayResize(slow,100000)
Test_ArrayResize (EURUSD,H1) 1. ArraySize(slow)=100000 Time=0 ms
Test_ArrayResize (EURUSD,H1) 2. ArraySize(slow)=200000 Time=0 ms
Test_ArrayResize (EURUSD,H1) 3. ArraySize(slow)=300000 Time=228511 ms
*/
```

See also[ArrayInitialize](#)

ArraySetAsSeries

The function sets the AS_SERIES flag to a selected [object of a dynamic array](#), and elements will be indexed like in [timeseries](#).

```
bool ArraySetAsSeries(
    const void& array[], // array by reference
    bool flag            // true denotes reverse order of indexing
);
```

Parameters

array[]

[in][out] Numeric array to set.

flag

[in] Array indexing direction.

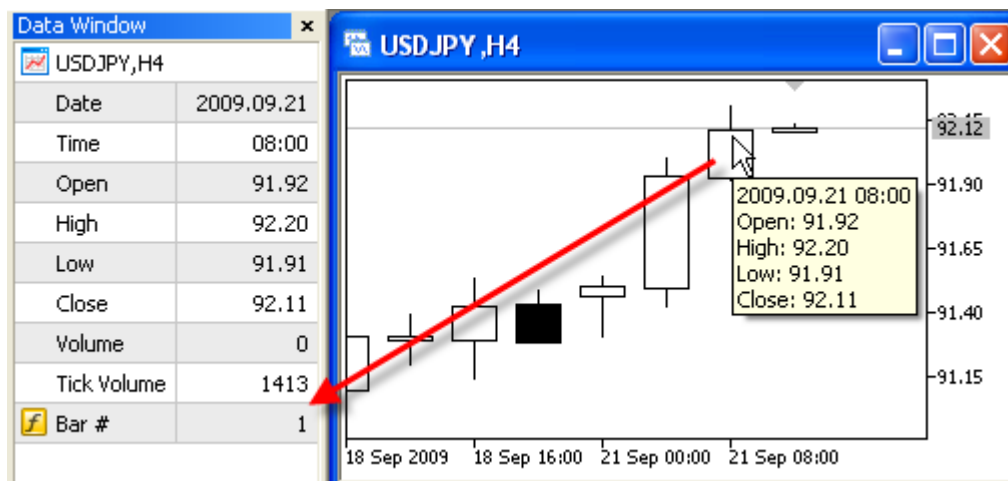
Return Value

The function returns true on success, otherwise - false.

Note

The [AS_SERIES](#) flag can't be set for multi-dimensional arrays or static arrays (arrays, whose size in square brackets is preset already on the compilation stage). Indexing in timeseries differs from a common array in that the elements of timeseries are indexed from the end towards the beginning (from the newest to oldest data).

Example: Indicator that shows bar number



```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Numeration
#property indicator_label1 "Numeration"
#property indicator_type1 DRAW_LINE
```



```

#property indicator_color1 CLR_NONE
//--- indicator buffers
double      NumerationBuffer[];
//+-----+
//| Custom indicator initialization function      |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- set indexing for the buffer like in timeseries
    ArraySetAsSeries(NumerationBuffer,true);
//--- set accuracy of showing in DataWindow
    IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- how the name of the indicator array is displayed in DataWindow
    PlotIndexSetString(0,PLOT_LABEL,"Bar #");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function          |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- we'll store the time of the current zero bar opening
    static datetime currentBarTimeOpen=0;
//--- revert access to array time[] - do it like in timeseries
    ArraySetAsSeries(time,true);
//--- If time of zero bar differs from the stored one
    if(currentBarTimeOpen!=time[0])
    {
//--- enumerate all bars from the current to the chart depth
        for(int i=rates_total-1;i>=0;i--) NumerationBuffer[i]=i;
        currentBarTimeOpen=time[0];
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

See also

[Access to timeseries](#), [ArrayGetAsSeries](#)

ArraySize

The function returns the number of elements of a selected array.

```
int ArraySize(
    const void& array[]    // checked array
);
```

Parameters

array[]
[in] Array of any type.

Return Value

Value of [int](#) type.

Note

For a one-dimensional array, the value to be returned by the [ArraySize](#) is equal to that of [ArrayRange](#)(array,0).

Example:

```
void OnStart()
{
    //--- create arrays
    double one_dim[];
    double four_dim[][10][5][2];
    //--- sizes
    int one_dim_size=25;
    int reserve=20;
    int four_dim_size=5;
    //--- auxiliary variable
    int size;
    //--- allocate memory without backup
    ArrayResize(one_dim,one_dim_size);
    ArrayResize(four_dim,four_dim_size);
    //--- 1. one-dimensional array
    Print("+=====+");
    Print("Array sizes:");
    Print("1. One-dimensional array");
    size=ArraySize(one_dim);
    PrintFormat("Zero dimension size = %d, Array size = %d",one_dim_size,size);
    //--- 2. multidimensional array
    Print("2. Multidimensional array");
    size=ArraySize(four_dim);
    PrintFormat("Zero dimension size = %d, Array size = %d",four_dim_size,size);
    //--- dimension sizes
    int d_1=ArrayRange(four_dim,1);
    int d_2=ArrayRange(four_dim,2);
    int d_3=ArrayRange(four_dim,3);
```



```
Print("Check:");
Print("Zero dimension = Array size / (First dimension * Second dimension * Third dimension)");
PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,d_3);
//--- 3. one-dimensional array with memory backup
Print("3. One-dimensional array with memory backup");
//--- double the value
one_dim_size*=2;
//--- allocate memory with backup
ArrayResize(one_dim,one_dim_size,reserve);
//--- print out the size
size=ArraySize(one_dim);
PrintFormat("Size with backup = %d, Actual array size = %d",one_dim_size,reserve,size);
}
```


ArraySort

Sorts the values in the first dimension of a multidimensional numeric array in the ascending order.

```
bool ArraySort(
    void& array[]    // array for sorting
);
```

Parameters

array[]
[in][out] Numeric array for sorting.

Return Value

The function returns true on success, otherwise - false.

Note

An array is always sorted in the ascending order irrespective of the [AS_SERIES](#) flag value.

Functions `ArraySort` and `ArrayBSearch` accept any-dimensional arrays as a parameter. However, searching and sorting are always applied to the first (zero) dimension.

Example:

```
#property description "The indicator analyzes data for the last month and draws all ca
#property description "and large tick volumes. The tick volume array is sorted out"
#property description "to define such candlesticks. The candlesticks having the volume
#property description "per cent of the array are considered small. The candlesticks ha
#property description "the last InpBigVolume per cent of the array are considered larg
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 1
//--- plot
#property indicator_label1 "VolumeFactor"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_color1 clrDodgerBlue,clrOrange
#property indicator_style1 STYLE_SOLID
#property indicator_width1 2
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input int InpSmallVolume=15; // Percentage value of small volumes (<50)
input int InpBigVolume=20; // Percentage value of large volumes (<50)
//--- analysis start time (will be shifted)
datetime ExtStartTime;
//--- indicator buffers
double ExtOpenBuff[];
double ExtHighBuff[];
double ExtLowBuff[];
```



```

double   ExtCloseBuff[];
double   ExtColorBuff[];
//--- volume boundary values for displaying the candlesticks
long     ExtLeftBorder=0;
long     ExtRightBorder=0;
//+-----+
//| Receive border values for tick volumes |
//+-----+
bool GetVolumeBorders(void)
{
//--- variables
    datetime stop_time; // copy end time
    long buff[]; // buffer for copying
//--- end time is the current one
    stop_time=TimeCurrent();
//--- start time is one month earlier from the current one
    ExtStartTime=GetStartTime(stop_time);
//--- receive the values of tick volumes
    ResetLastError();
    if(CopyTickVolume(Symbol(),Period(),ExtStartTime,stop_time,buff)==-1)
    {
        //--- failed to receive the data, return false to launch recalculation command
        PrintFormat("Failed to receive tick volume values. Error code = %d",GetLastError());
        return(false);
    }
//--- calculate array size
    int size=ArraySize(buff);
//--- sort out the array
    ArraySort(buff);
//--- define the values of the left and right border for tick volumes
    ExtLeftBorder=buff[size*InpSmallVolume/100];
    ExtRightBorder=buff[(size-1)*(100-InpBigVolume)/100];
//--- successful execution
    return(true);
}
//+-----+
//| Receive the data that is one month less than the passed one |
//+-----+
datetime GetStartTime(const datetime stop_time)
{
//--- convert end time into MqlDateTime type structure variable
    MqlDateTime temp;
    TimeToStruct(stop_time,temp);
//--- receive the data that is one month less
    if(temp.mon>1)
        temp.mon-=1; // the current month is not the first one in the year, therefore,
    else
    {
        temp.mon=12; // the current month is the first in the year, therefore, the num

```



```

        temp.year--; // while the year number is one less
    }
    //--- day number will not exceed 28
    if(temp.day>28)
        temp.day=28;
    //--- return the obtained date
    return(StructToTime(temp));
}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- check if input parameters satisfy the conditions
    if(InpSmallVolume<0 || InpSmallVolume>=50 || InpBigVolume<0 || InpBigVolume>=50)
    {
        Print("Incorrect input parameters");
        return(INIT_PARAMETERS_INCORRECT);
    }
    //--- indicator buffers mapping
    SetIndexBuffer(0,ExtOpenBuff);
    SetIndexBuffer(1,ExtHighBuff);
    SetIndexBuffer(2,ExtLowBuff);
    SetIndexBuffer(3,ExtCloseBuff);
    SetIndexBuffer(4,ExtColorBuff,INDICATOR_COLOR_INDEX);
    //--- set the value that will not be displayed
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
    //--- set labels for indicator buffers
    PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- check if unhandled bars are still present
    if(prev_calculated<rates_total)
    {

```



```

    //--- receive new values of the right and left borders for volumes
    if(!GetVolumeBorders())
        return(0);
}

//--- start variable for bar calculation
int start=prev_calculated;
//--- work at the last bar if the indicator values have already been calculated at the
if(start>0)
    start--;
//--- set direct indexing in time series
ArraySetAsSeries(time,false);
ArraySetAsSeries(open,false);
ArraySetAsSeries(high,false);
ArraySetAsSeries(low,false);
ArraySetAsSeries(close,false);
ArraySetAsSeries(tick_volume,false);
//--- the loop of calculation of the indicator values
for(int i=start;i<rates_total;i++)
{
    //--- fill out candlesticks starting from the initial date
    if(ExtStartTime<=time[i])
    {
        //--- if the value is not less than the right border, fill out the candlestick
        if(tick_volume[i]>=ExtRightBorder)
        {
            //--- receive data for drawing the candlestick
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- DodgerBlue color
            ExtColorBuff[i]=0;
            //--- continue the loop
            continue;
        }
        //--- fill out the candlestick if the value does not exceed the left border
        if(tick_volume[i]<=ExtLeftBorder)
        {
            //--- receive data for drawing the candlestick
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- Orange color
            ExtColorBuff[i]=1;
            //--- continue the loop
            continue;
        }
    }
}

```



```
//--- set empty values for bars that have not been included in the calculation
ExtOpenBuff[i]=INDICATOR_EMPTY_VALUE;
ExtHighBuff[i]=INDICATOR_EMPTY_VALUE;
ExtLowBuff[i]=INDICATOR_EMPTY_VALUE;
ExtCloseBuff[i]=INDICATOR_EMPTY_VALUE;
}
//--- return value of prev_calculated for next call
return(rates_total);
}
```

See also[ArrayBsearch](#)

Conversion Functions

This is a group of functions that provide conversion of data from one format into another.

The [NormalizeDouble\(\)](#) function must be specially noted as it provides the necessary accuracy of the price presentation. In trading operations, no unnormalized prices may be used if their accuracy even a digit exceeds that required by the trade server.

Function	Action
CharToString	Converting a symbol code into a one-character string
DoubleToString	Converting a numeric value to a text line with a specified accuracy
EnumToString	Converting an enumeration value of any type to string
NormalizeDouble	Rounding of a floating point number to a specified accuracy
StringToDouble	Converting a string containing a symbol representation of number into number of double type
StringToInteger	Converting a string containing a symbol representation of number into number of int type
StringToTime	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type
TimeToString	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format
IntegerToString	Converting int into a string of preset length
ShortToString	Converting symbol code (unicode) into one-symbol string
ShortArrayToString	Copying array part into a string
StringToShortArray	Symbol-wise copying a string to a selected part of array of ushort type
CharArrayToString	Converting symbol code (ansi) into one-symbol array
StringToCharArray	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type

<u>ColorToARGB</u>	Converting color type to uint type to receive ARGB representation of the color.
<u>ColorToString</u>	Converting color value into string as "R,G,B"
<u>StringToColor</u>	Converting "R,G,B" string or string with color name into color type value
<u>StringFormat</u>	Converting number into string according to preset format

See also

[Use of a Codepage](#)

CharToString

Converting a symbol code into a one-character string.

```
string CharToString(  
    uchar char_code    // numeric code of symbol  
);
```

Parameters

char_code

[in] Code of ANSI symbol.

Return Value

String with a ANSI symbol.

CharArrayToString

It copies and converts part of array of uchar type into a returned string.

```
string CharArrayToString(  
    uchar array[],           // array  
    int start=0,             // starting position in the array  
    int count=-1             // number of symbols  
    uint codepage=CP_ACP     // code page  
);
```

Parameters

array[]

[in] Array of uchar type.

start=0

[in] Position from which copying starts. by default 0 is used.

count=-1

[in] Number of array elements for copying. Defines the length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.

codepage=CP_ACP

[in] The value of the code page. There is a number of built-in constants for the most used [code pages](#).

Return Value

String.

See also

[Use of a Codepage](#)

ColorToARGB

The function converts [color](#) type into [uint](#) type to get ARGB representation of the color. ARGB color format is used to generate a [graphical resource](#), [text display](#), as well as for CCanvas standard library class.

```
uint ColorToARGB (
    color clr,           // converted color in color format
    uchar alpha=255     // alpha channel managing color transparency
);
```

Parameters

clr

[in] Color value in color type variable.

alpha

[in] The value of the alpha channel used to receive the color in [ARGB](#) format. The value may be set from 0 (a color of a foreground pixel does not change the display of an underlying one) up to 255 (a color of an underlying pixel is completely replaced by the foreground pixel's one). Color transparency in percentage terms is calculated as $(1 - \alpha / 255) * 100\%$. In other words, the lesser value of the alpha channel leads to more transparent color.

Return Value

Presenting the color in ARGB format where Alfa, Red, Green, Blue (alpha channel, red, green, blue) values are set in series in four uint type bytes.

Note

RGB is a basic and commonly used format for pixel color description on a screen in computer graphics. Names of basic colors are used to set red, green and blue color components. Each component is described by one byte specifying the color saturation in the range of 0 to 255 (0x00 to 0xFF in hexadecimal format). Since the white color contains all colors, it is described as 0xFFFFFF, that is, each one of three components is presented by the maximum value of 0xFF.

However, some tasks require to specify the color transparency to describe the look of an image in case it is covered by the color with some degree of transparency. The concept of alpha channel is introduced for such cases. It is implemented as an additional component of RGB format. ARGB format structure is shown below.

8								8								8								8							
Alpha								Red								Green								Blue							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ARGB values are typically expressed using hexadecimal format with each pair of digits representing the values of Alpha, Red, Green and Blue channels, respectively. For example, 80FFFF00 color represents 50.2% opaque yellow. Initially, 0x80 sets 50.2% alpha value, as it is 50.2% of 0xFF value. Then, the first FF pair defines the highest value of the red component; the next FF pair is like the previous but for the green component; the final 00 pair represents the lowest value the blue component can have (absence of blue). Combination of green and red colors yields yellow one. If the

alpha channel is not used, the entry can be reduced down to 6 RRGGBB digits, this is why the alpha channel values are stored in the top bits of uint integer type.

Depending on the context, hexadecimal digits can be written with '0x' or '#' prefix, for example, 80FFFF00, 0x80FFFF00 or #80FFFF00.

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- set transparency
uchar alfa=0x55; // 0x55 means 55/255=21.6 % of transparency
//--- derive conversion to ARGB for clrBlue color
PrintFormat("0x%.8X - clrBlue",clrBlue);
PrintFormat("0x%.8X - clrBlue ARGB with alfa=0x55 (transparency 21.6%)",ColorToARGB(clrBlue,alfa));
//--- derive conversion to ARGB for clrGreen color
PrintFormat("0x%.8X - clrGreen",clrGreen);
PrintFormat("0x%.8X - clrGreen ARGB with alfa=0x55 (transparency 21.6%)",ColorToARGB(clrGreen,alfa));
//--- derive conversion to ARGB for clrRed color
PrintFormat("0x%.8X - clrRed",clrRed);
PrintFormat("0x%.8X - clrRed ARGB with alfa=0x55 (transparency 21.6%)",ColorToARGB(clrRed,alfa));
}
```

See also

[Resources](#), [ResourceCreate\(\)](#), [TextOut\(\)](#), [color type](#), [char](#), [short](#), [int](#) and [long types](#)

ColorToString

It converts color value into string of "R,G,B" form.

```
string ColorToString(  
    color  color_value,      // color value  
    bool   color_name       // show color name or not  
);
```

Parameters

color_value

[in] Color value in color type variable.

color_name

[in] Return color name if it is identical to one of predefined [color constants](#).

Return Value

String presentation of color as "R,G,B", where R, G and B are decimal constants from 0 to 255 converted into a string. If the color_name=true parameter is set, it will try to convert color value into color name.

Example:

```
string clr=ColorToString(C'0,255,0'); // green color  
Print(clr);  
  
clr=ColorToString(C'0,255,0',true);   // get color constant  
Print(clr);
```


DoubleToString

Converting numeric value into text string.

```
string DoubleToString(  
    double value,      // number  
    int digits=8       // number of digits after decimal point  
);
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format. If the *digits* value is in the range between 0 and 16, a string presentation of a number with the specified number of digits after the point will be obtained. If the *digits* value is in the range between -1 and -16, a string representation of a number in the scientific format with the specified number of digits after the decimal point will be obtained. In all other cases the string value will contain 8 digits after the decimal point.

Return Value

String containing a symbol representation of a number with the specified accuracy.

Example:

```
Print("DoubleToString(120.0 + M_PI) : ",DoubleToString(120.0+M_PI));  
Print("DoubleToString(120.0 + M_PI,16) : ",DoubleToString(120.0+M_PI,16));  
Print("DoubleToString(120.0 + M_PI,-16) : ",DoubleToString(120.0+M_PI,-16));  
Print("DoubleToString(120.0 + M_PI,-1) : ",DoubleToString(120.0+M_PI,-1));  
Print("DoubleToString(120.0 + M_PI,-20) : ",DoubleToString(120.0+M_PI,-20));
```

See also

[NormalizeDouble](#), [StringToDouble](#)

EnumToString

Converting an enumeration value of any type to a text form.

```
string EnumToString(
    any_enum value    // any type enumeration value
);
```

Parameters

value

[in] Any type enumeration value.

Return Value

A string with a text representation of the enumeration. To get the error message call the [GetLastError\(\)](#) function.

Note

The function can set the following error values in the [_LastError](#) variable:

- ERR_INTERNAL_ERROR - error of the execution environment
- ERR_NOT_ENOUGH_MEMORY - not enough memory to complete the operation
- ERR_INVALID_PARAMETER - can't allow the name of the enumeration value

Example:

```
enum interval // enumeration of named constants
{
    month=1,      // one-month interval
    two_months,   // two months
    quarter,      // three months - a quarter
    halfyear=6,   // half a year
    year=12,      // a year - 12 months
};

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- set the time interval equal to one month
    interval period=month;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- set the time interval equal to a quarter (three months)
    period=quarter;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- set the time interval equal to one year (12 months)
    period=year;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- check how the order type is shown
```



```
ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;
Print(EnumToString(type)+"="+IntegerToString(type));

//--- check how incorrect values are shown
type=WRONG_VALUE;
Print(EnumToString(type)+"="+IntegerToString(type));

// Result:
// month=1
// quarter=3
// year=12
// ORDER_TYPE_BUY=0
// ENUM_ORDER_TYPE:: -1=-1
}
```

See also

[Enumerations](#), [Input variables](#)

IntegerToString

This function converts value of integer type into a string of a specified length and returns the obtained string.

```
string IntegerToString(  
    long    number,           // number  
    int     str_len=0,       // length of result string  
    ushort  fill_symbol=' '  // filler  
);
```

Parameters

number

[in] Number for conversion.

str_len=0

[in] String length. If the resulting string length is larger than the specified one, the string is not cut off. If it is smaller, filler symbols will be added to the left.

fill_symbol=' '

[in] Filler symbol. By default it is a space.

Return Value

String.

ShortToString

It converts the symbol code (unicode) into one-symbol string and returns resulting string.

```
string ShortToString(  
    ushort symbol_code    // symbol  
);
```

Parameters

symbol_code

[in] Symbol code. Instead of a symbol code you can use literal string containing a symbol or a literal string with 2-byte hexadecimal code corresponding to the symbol from the Unicode table.

Return Value

String.

ShortArrayToString

It copies part of array into a returned string.

```
string ShortArrayToString(  
    ushort array[],      // array  
    int start=0,         // starting position in the array  
    int count=-1         // number of symbols  
);
```

Parameters

array[]

[in] Array of ushort type (analog of wchar_t type).

start=0

[in] Position, from which copying starts, Default - 0.

count=-1

[in] Number of array elements to copy. Defines the length of a resulting string. Default value is - 1, which means copying up to the array end, or till terminal 0.

Return Value

String.

TimeToString

Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyymm.dd hh:mi" format.

```
string TimeToString(  
    datetime value,                // number  
    int mode=TIME_DATE|TIME_MINUTES // output format  
);
```

Parameters

value

[in] Time in seconds from 00:00 1970/01/01.

mode=TIME_DATE|TIME_MINUTES

[in] Additional data input mode. Can be one or combined flag:

TIME_DATE gets result as "yyyymm.dd",

TIME_MINUTES gets result as "hh:mi",

TIME_SECONDS gets results as "hh:mi:ss".

Return Value

String.

NormalizeDouble

Rounding floating point number to a specified accuracy.

```
double NormalizeDouble(
    double value,      // normalized number
    int    digits      // number of digits after decimal point
);
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format, number of digits after point (0-8).

Return Value

Value of double type with preset accuracy.

Note

Calculated values of StopLoss, TakeProfit, and values of open prices for pending orders must be normalized with the accuracy, the value of which can be obtained by [Digits\(\)](#).

Please note that when output to Journal using the Print() function, a normalized number may contain a greater number of decimal places than you expect. For example, for:

```
double a=76.671;           // A normalized number with three decimal places
Print("Print(76.671)=",a); // Output as is
Print("DoubleToString(a,8)=",DoubleToString(a,8)); // Output with a preset accuracy
```

you will have the following in the terminal:

```
DoubleToString(a,8)=76.67100000
```

```
Print(76.671)=76.671000000000001
```

Example:

```
double pi=M_PI;
Print("pi = ",DoubleToString(pi,16));

double pi_3=NormalizeDouble(M_PI,3);
Print("NormalizeDouble(pi,3) = ",DoubleToString(pi_3,16))
;
double pi_8=NormalizeDouble(M_PI,8);
Print("NormalizeDouble(pi,8) = ",DoubleToString(pi_8,16));

double pi_0=NormalizeDouble(M_PI,0);
Print("NormalizeDouble(pi,0) = ",DoubleToString(pi_0,16));
/*
Result:
```



```
pi= 3.1415926535897931
NormalizeDouble(pi,3)= 3.1419999999999999
NormalizeDouble(pi,8)= 3.1415926499999998
NormalizeDouble(pi,0)= 3.0000000000000000
*/
```

See also

[DoubleToString](#), [Real types \(double, float\)](#), [Reduction of types](#),

StringToCharArray

Symbol-wise copies a string converted from Unicode to ANSI, to a selected place of array of uchar type. It returns the number of copied elements.

```
int StringToCharArray(  
    string text_string,           // source string  
    uchar& array[],              // array  
    int start=0,                  // starting position in the array  
    int count=-1                  // number of symbols  
    uint codepage=CP_ACP         // code page  
);
```

Parameters

text_string

[in] String to copy.

array[]

[out] Array of uchar type.

start=0

[in] Position from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0. Terminal 0 will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

codepage=CP_ACP

[in] The value of the code page. For the most-used [code pages](#) provide appropriate constants.

Return Value

Number of copied elements.

See also

[Use of a Codepage](#)

StringToColor

Converting "R,G,B" string or string with color name into color type value.

```
color StringToColor(  
    string color_string    // string representation of color  
);
```

Parameters

color_string

[in] String representation of a color of "R,G,B" type or name of one of predefined [Web-colors](#).

Return Value

Color value.

Example:

```
color str_color=StringToColor("0,127,0");  
Print(str_color);  
Print((string)str_color);  
//--- change color a little  
str_color=StringToColor("0,128,0");  
Print(str_color);  
Print((string)str_color);
```


StringToDouble

The function converts string containing a symbol representation of number into number of double type.

```
double StringToDouble(  
    string value    // string  
);
```

Parameters

value

[in] String containing a symbol representation of a number.

Return Value

Value of double type.

StringToInteger

The function converts string containing a symbol representation of number into number of int (integer) type.

```
long StringToInteger(  
    string value    // string  
);
```

Parameters

value

[in] String containing a number.

Return Value

Value of long type.

StringToShortArray

The function symbol-wise copies a string into a specified place of an array of ushort type. It returns the number of copied elements.

```
int StringToShortArray(  
    string text_string,      // source string  
    ushort& array[],        // array  
    int start=0,            // starting position in the array  
    int count=-1            // number of symbols  
);
```

Parameters

text_string

[in] String to copy

array[]

[out] Array of [ushort](#) type (analog of wchar_t type).

start=0

[in] Position, from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0. Terminal 0 will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

Return Value

Number of copied elements.

StringToTime

The function converts a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type.

```
datetime StringToTime(  
    string value    // date string  
);
```

Parameters

value

[in] String in " yyyy.mm.dd hh:mi " format.

Return Value

Value of [datetime](#) type containing total number of seconds that elapsed since 01.01.1970.

StringFormat

The function formats obtained parameters and returns a string.

```
string StringFormat(  
    string format,      // string with format description  
    ...               // parameters  
);
```

Parameters

format

[in] String containing method of formatting. Formatting rules are the same as for the [PrintFormat](#) function.

...

[in] Parameters, separated by a comma.

Return Value

String.

Example:


```

void OnStart()
{
//--- string variables
    string output_string;
    string temp_string;
    string format_string;
//--- prepare the specification header
    temp_string=StringFormat("Contract specification for %s:\n",_Symbol);
    StringAdd(output_string,temp_string);
//--- int value output
    int digits=(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
    temp_string=StringFormat("    SYMBOL_DIGITS = %d (number of digits after the decimal
        digits);
    StringAdd(output_string,temp_string);
//--- double value output with variable number of digits after the decimal point
    double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
    format_string=StringFormat("    SYMBOL_POINT = %%.%df (point value)\n",
        digits);
    temp_string=StringFormat(format_string,point_value);
    StringAdd(output_string,temp_string);
//--- int value output
    int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
    temp_string=StringFormat("    SYMBOL_SPREAD = %d (current spread in points)\n",
        spread);
    StringAdd(output_string,temp_string);
//--- int value output
    int min_stop=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
    temp_string=StringFormat("    SYMBOL_TRADE_STOPS_LEVEL = %d (minimal indention in p
        min_stop);
    StringAdd(output_string,temp_string);
//--- double value output without the fractional part
    double contract_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_CONTRACT_SIZE);
    temp_string=StringFormat("    SYMBOL_TRADE_CONTRACT_SIZE = %.f (contract size)\n",
        contract_size);
    StringAdd(output_string,temp_string);
//--- double value output with default accuracy
    double tick_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_TICK_SIZE);
    temp_string=StringFormat("    SYMBOL_TRADE_TICK_SIZE = %f (minimal price change)\n",
        tick_size);
    StringAdd(output_string,temp_string);
//--- determining the swap calculation mode
    int swap_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_SWAP_MODE);
    string str_swap_mode;
    switch(swap_mode)
    {
        case SYMBOL_SWAP_MODE_DISABLED: str_swap_mode="SYMBOL_SWAP_MODE_DISABLED (no swa
        case SYMBOL_SWAP_MODE_POINTS: str_swap_mode="SYMBOL_SWAP_MODE_POINTS (in points)
        case SYMBOL_SWAP_MODE_CURRENCY_SYMBOL: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
        case SYMBOL_SWAP_MODE_CURRENCY_MARGIN: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
        case SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
        case SYMBOL_SWAP_MODE_INTEREST_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST
        case SYMBOL_SWAP_MODE_INTEREST_OPEN: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST_O
        case SYMBOL_SWAP_MODE_REOPEN_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_CU
        case SYMBOL_SWAP_MODE_REOPEN_BID: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_BID (by
    }
//--- string value output
    temp_string=StringFormat("    SYMBOL_SWAP_MODE = %s\n",
        str_swap_mode);
    StringAdd(output_string,temp_string);
//--- double value output with default accuracy
    double swap_long=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_LONG);

```



```

temp_string=StringFormat("    SYMBOL_SWAP_LONG = %f (long swap value)\n",
                        swap_long);
StringAdd(output_string,temp_string);
//--- double value output with default accuracy
double swap_short=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_SHORT);
temp_string=StringFormat("    SYMBOL_SWAP_SHORT = %f (short swap value)\n",
                        swap_short);
StringAdd(output_string,temp_string);
//--- determining the trading mode
int trade_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_MODE);
string str_trade_mode;
switch(trade_mode)
{
    case SYMBOL_TRADE_MODE_DISABLED: str_trade_mode="SYMBOL_TRADE_MODE_DISABLED (trading disabled)";
    case SYMBOL_TRADE_MODE_LONGONLY: str_trade_mode="SYMBOL_TRADE_MODE_LONGONLY (only long trades)";
    case SYMBOL_TRADE_MODE_SHORTONLY: str_trade_mode="SYMBOL_TRADE_MODE_SHORTONLY (only short trades)";
    case SYMBOL_TRADE_MODE_CLOSEONLY: str_trade_mode="SYMBOL_TRADE_MODE_CLOSEONLY (only close trades)";
    case SYMBOL_TRADE_MODE_FULL: str_trade_mode="SYMBOL_TRADE_MODE_FULL (no trade restrictions)";
}
//--- string value output
temp_string=StringFormat("    SYMBOL_TRADE_MODE = %s\n",
                        str_trade_mode);
StringAdd(output_string,temp_string);
//--- double value output in a compact format
double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
temp_string=StringFormat("    SYMBOL_VOLUME_MIN = %g (minimal volume for a deal)\n",
                        volume_min);
StringAdd(output_string,temp_string);
//--- double value output in a compact format
double volume_step=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_STEP);
temp_string=StringFormat("    SYMBOL_VOLUME_STEP = %g (minimal volume change step)\n",
                        volume_step);
StringAdd(output_string,temp_string);
//--- double value output in a compact format
double volume_max=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MAX);
temp_string=StringFormat("    SYMBOL_VOLUME_MAX = %g (maximal volume for a deal)\n",
                        volume_max);
StringAdd(output_string,temp_string);
//--- determining the contract price calculation mode
int calc_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_CALC_MODE);
string str_calc_mode;
switch(calc_mode)
{
    case SYMBOL_CALC_MODE_FOREX: str_calc_mode="SYMBOL_CALC_MODE_FOREX (Forex)";break;
    case SYMBOL_CALC_MODE_FUTURES: str_calc_mode="SYMBOL_CALC_MODE_FUTURES (futures)";break;
    case SYMBOL_CALC_MODE_CFD: str_calc_mode="SYMBOL_CALC_MODE_CFD (CFD)";break;
    case SYMBOL_CALC_MODE_CFDINDEX: str_calc_mode="SYMBOL_CALC_MODE_CFDINDEX (CFD for index)";break;
    case SYMBOL_CALC_MODE_CFDLEVERAGE: str_calc_mode="SYMBOL_CALC_MODE_CFDLEVERAGE (CFD for leverage)";break;
    case SYMBOL_CALC_MODE_EXCH_STOCKS: str_calc_mode="SYMBOL_CALC_MODE_EXCH_STOCKS (exchange stocks)";break;
    case SYMBOL_CALC_MODE_EXCH_FUTURES: str_calc_mode="SYMBOL_CALC_MODE_EXCH_FUTURES (exchange futures)";break;
    case SYMBOL_CALC_MODE_EXCH_FORTS: str_calc_mode="SYMBOL_CALC_MODE_EXCH_FORTS (exchange forts)";break;
}
//--- string value output
temp_string=StringFormat("    SYMBOL_TRADE_CALC_MODE = %s\n",
                        str_calc_mode);
StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
double margin_initial=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_INITIAL);
temp_string=StringFormat("    SYMBOL_MARGIN_INITIAL = %.2f (initial margin)\n",
                        margin_initial);
StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
double margin_maintenance=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_MAINTENANCE);
temp_string=StringFormat("    SYMBOL_MARGIN_MAINTENANCE = %.2f (maintenance margin)\n",
                        margin_maintenance);
StringAdd(output_string,temp_string);

```



```

        margin_maintenance);
StringAdd(output_string,temp_string);
//--- int value output
int freeze_level=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_FREEZE_LEVEL);
temp_string=StringFormat("    SYMBOL_TRADE_FREEZE_LEVEL = %d (order freeze level in
        freeze_level);
StringAdd(output_string,temp_string);
Print(output_string);
Comment(output_string);
/* execution result
Contract specification for EURUSD:
    SYMBOL_DIGITS = 5 (number of digits after the decimal point)
    SYMBOL_POINT = 0.00001 (point value)
    SYMBOL_SPREAD = 10 (current spread in points)
    SYMBOL_TRADE_STOPS_LEVEL = 18 (minimal indention in points for Stop orders)
    SYMBOL_TRADE_CONTRACT_SIZE = 100000 (contract size)
    SYMBOL_TRADE_TICK_SIZE = 0.000010 (minimal price change)
    SYMBOL_SWAP_MODE = SYMBOL_SWAP_MODE_POINTS (in points)
    SYMBOL_SWAP_LONG = -0.700000 (buy order swap value)
    SYMBOL_SWAP_SHORT = -1.000000 (sell order swap value)
    SYMBOL_TRADE_MODE = SYMBOL_TRADE_MODE_FULL (no trade restrictions)
    SYMBOL_VOLUME_MIN = 0.01 (minimal volume for a deal)
    SYMBOL_VOLUME_STEP = 0.01 (minimal volume change step)
    SYMBOL_VOLUME_MAX = 500 (maximal volume for a deal)
    SYMBOL_TRADE_CALC_MODE = SYMBOL_CALC_MODE_FOREX (Forex)
    SYMBOL_MARGIN_INITIAL = 0.00 (initial margin)
    SYMBOL_MARGIN_MAINTENANCE = 0.00 (maintenance margin)
    SYMBOL_TRADE_FREEZE_LEVEL = 0 (order freeze level in points)
*/
}

```

See also

[PrintFormat](#), [DoubleToString](#), [ColorToString](#), [TimeToString](#)

Mathematical Functions

A set of mathematical and trigonometric functions.

Function	Action
<u>MathAbs</u>	Returns absolute value (modulus) of the specified numeric value
<u>MathArccos</u>	Returns the arc cosine of x in radians
<u>MathArcsin</u>	Returns the arc sine of x in radians
<u>MathArctan</u>	Returns the arc tangent of x in radians
<u>MathCeil</u>	Returns integer numeric value closest from above
<u>MathCos</u>	Returns the cosine of a number
<u>MathExp</u>	Returns exponent of a number
<u>MathFloor</u>	Returns integer numeric value closest from below
<u>MathLog</u>	Returns natural logarithm
<u>MathLog10</u>	Returns the logarithm of a number by base 10
<u>MathMax</u>	Returns the maximal value of the two numeric values
<u>MathMin</u>	Returns the minimal value of the two numeric values
<u>MathMod</u>	Returns the real remainder after the division of two numbers
<u>MathPow</u>	Raises the base to the specified power
<u>MathRand</u>	Returns a pseudorandom value within the range of 0 to 32767
<u>MathRound</u>	Rounds of a value to the nearest integer
<u>MathSin</u>	Returns the sine of a number
<u>MathSqrt</u>	Returns a square root
<u>MathSrand</u>	Sets the starting point for generating a series of pseudorandom integers
<u>MathTan</u>	Returns the tangent of a number
<u>MathIsValidNumber</u>	Checks the correctness of a real number
<u>MathExpM1</u>	Returns the value of the expression $\text{MathExp}(x)-1$

<u>MathLog1p</u>	Returns the value of the expression $\text{MathLog}(1+x)$
<u>MathArccosh</u>	Returns the hyperbolic arccosine
<u>MathArcsinh</u>	Returns the hyperbolic arcsine
<u>MathArctanh</u>	Returns the hyperbolic arctangent
<u>MathCosh</u>	Returns the hyperbolic cosine
<u>MathSinh</u>	Returns the hyperbolic sine
<u>MathTanh</u>	Returns the hyperbolic tangent

MathAbs

The function returns the absolute value (modulus) of the specified numeric value.

```
double MathAbs(  
    double value    // numeric value  
);
```

Parameters

value

[in] Numeric value.

Return Value

Value of double type more than or equal to zero.

Note

Instead the MathAbs() function you can use [fabs\(\)](#).

MathArccos

The function returns the arccosine of x within the range 0 to π in radians.

```
double MathArccos(  
    double val    // -1<val<1  
) ;
```

Parameters

val

[in] The val value between -1 and 1, the arc cosine of which is to be calculated.

Return Value

Arc cosine of a number in radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

Instead of the MathArccos() function you can use [acos\(\)](#).

See also

[Real types \(double, float\)](#)

MathArcsin

The function returns the arc sine of x within the range of $-\pi/2$ to $\pi/2$ radians.

```
double MathArcsin(  
    double val      // -1<value<1  
);
```

Parameters

val

[in] The val value between -1 and 1, the arc sine of which is to be calculated.

Return Value

Arc sine of the val number in radians within the range of $-\pi/2$ to $\pi/2$ radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

Instead of the MathArcsin() function you can use [asin\(\)](#).

See also

[Real types \(double, float\)](#)

MathArctan

The function returns the arc tangent of x. If x is equal to 0, the function returns 0.

```
double MathArctan(  
    double value    // tangent  
);
```

Parameters

value

[in] A number representing a tangent.

Return Value

MathArctan returns a value within the range of $-\pi/2$ to $\pi/2$ radians.

Note

Instead of the MathArctan() function you can use [atan\(\)](#).

MathCeil

The function returns integer numeric value closest from above.

```
double MathCeil(  
    double val    // number  
);
```

Parameters

val

[in] Numeric value.

Return Value

Numeric value representing the smallest integer that exceeds or equals to val.

Note

Instead of the MathCeil() function you can use [ceil\(\)](#).

MathCos

The function returns the cosine of an angle.

```
double MathCos(  
    double value    // number  
);
```

Parameters

value

[in] Angle in radians.

Return Value

Value of double type within the range of -1 to 1.

Note

Instead of MathCos() you can use [cos\(\)](#).

MathExp

The function returns the value of e raised to the power of d.

```
double MathExp(  
    double value    // power for the number e  
);
```

Parameters

value

[in] A number specifying the power.

Return Value

A number of double type. In case of overflow the function returns INF (infinity), in case of underflow MathExp returns 0.

Note

Instead of MathExp() you can use [exp\(\)](#).

See also

[Real types \(double, float\)](#)

MathFloor

The function returns integer numeric value closest from below.

```
double MathFloor(  
    double val    // number  
);
```

Parameters

val

[in] Numeric value.

Return Value

A numeric value representing the largest integer that is less than or equal to val.

Note

Instead of MathFloor() you can use [floor\(\)](#).

MathLog

The function returns a natural logarithm.

```
double MathLog(  
    double val    // value to take the logarithm  
);
```

Parameters

val

[in] Value logarithm of which is to be found.

Return Value

The natural logarithm of *val* in case of success. If *val* is negative, the function returns NaN (undetermined value). If *val* is equal to 0, the function returns INF (infinity).

Note

Instead of `MathLog()` you can use `log()`.

See also

[Real types \(double, float\)](#)

MathLog

Returns the logarithm of a number by base 10.

```
double MathLog10(  
    double val    // number to take logarithm  
);
```

Parameters

val

[in] Numeric value the common logarithm of which is to be calculated.

Return Value

The common logarithm in case of success. If *val* is negative, the function returns NaN (undetermined value). If *val* is equal to 0, the function returns INF (infinity).

Note

Instead of MathLog10() you can use [log10\(\)](#).

See also

[Real types \(double, float\)](#)

MathMax

The function returns the maximal value of two values.

```
double MathMax(  
    double value1,    // first value  
    double value2     // second value  
);
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The largest of the two values.

Note

Instead of `MathMax()` you can use `fmax()`. Functions `fmax()`, `fmin()`, `MathMax()`, `MathMin()` can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically cast to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

MathMin

The function returns the minimal value of two values.

```
double MathMin(  
    double value1,    // first value  
    double value2     // second value  
);
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The smallest of the two values.

Note

Instead of MathMin() you can use [fmin\(\)](#). Functions [fmax\(\)](#), [fmin\(\)](#), [MathMax\(\)](#), [MathMin\(\)](#) can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically [cast](#) to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

MathMod

The function returns the real remainder of division of two numbers.

```
double MathMod(  
    double value,      // dividend value  
    double value2      // divisor value  
);
```

Parameters

value

[in] Dividend value.

value2

[in] Divisor value.

Return Value

The MathMod function calculates the real remainder f from expression val/y so that $val = i * y + f$, where i is an integer, f has the same sign as val , and the absolute value of f is less than the absolute value of y .

Note

Instead of MathMod() you can use [fmod\(\)](#).

MathPow

The function raises a base to a specified power.

```
double MathPow(  
    double base,           // base  
    double exponent        // exponent value  
);
```

Parameters

base

[in] Base.

exponent

[in] Exponent value.

Return Value

Value of base raised to the specified power.

Note

Instead of MathPow() you can use [pow\(\)](#).

MathRand

Returns a pseudorandom integer within the range of 0 to 32767.

```
int MathRand();
```

Return Value

Integer value within the range of 0 to 32767.

Note

Before the first call of the function, it's necessary to call [MathSrand](#) to set the generator of pseudorandom numbers to the initial state.

Note

Instead of MathRand() you can use [rand\(\)](#).

MathRound

The function returns a value rounded off to the nearest integer of the specified numeric value.

```
double MathRound(  
    double value    // value to be rounded  
);
```

Parameters

value

[in] Numeric value before rounding.

Return Value

Value rounded till to the nearest integer.

Note

Instead of MathRound() you can use [round\(\)](#).

MathSin

Returns the sine of a specified angle.

```
double MathSin(  
    double value    // argument in radians  
);
```

Parameters

value

[in] Angle in radians.

Return Value

Sine of an angle measured in radians. Returns value within the range of -1 to 1.

Note

Instead of MathSin() you can use [sin\(\)](#).

MathSqrt

Returns the square root of a number.

```
double MathSqrt(  
    double value    // positive number  
);
```

Parameters

value

[in] Positive numeric value.

Return Value

Square root of value. If value is negative, MathSqrt returns NaN (indeterminate value).

Note

Instead of MathSqrt() you can use [sqrt\(\)](#).

See also

[Real types \(double, float\)](#)

MathSrand

Sets the starting point for generating a series of pseudorandom integers.

```
void MathSrand(
    int seed    // initializing number
);
```

Parameters

seed

[in] Starting number for the sequence of random numbers.

Return Value

No return value.

Note

The [MathRand\(\)](#) function is used for generating a sequence of pseudorandom numbers. Call of MathSrand() with a certain initializing number allows to always produces the same sequence of pseudorandom numbers.

To ensure receipt of non-recurring sequence, use the call of MathSrand(GetTickCount()), since the value of [GetTickCount\(\)](#) increases from the moment of the start of the operating system and is not repeated within 49 days, until the built-in counter of milliseconds overflows. Use of MathSrand(TimeCurrent()) is not suitable, because the [TimeCurrent\(\)](#) function returns the time of the last tick, which can be unchanged for a long time, for example at the weekend.

Initialization of the random number generator using MathSrand() for indicators and Expert Advisors is better performed in the OnInit() handler; it saves you from the following multiple restarts of the generator in OnTick() and OnCalculate().

Instead of the MathSrand() function you can use the [srand\(\)](#) function.

Example:

```
#property description "The indicator shows the central limit theorem, which states:"
#property description "The sum of a sufficiently large number of weakly dependent random variables,
#property description "having approximately equal magnitude (none of the summands dominates),
#property description "or makes a determining contribution to the sum), has a distribution that approaches a normal distribution."

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- Properties of the graphical construction
#property indicator_label1 "Label"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 clrRoyalBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 5
//--- An input variable
input int sample_number=10;
//--- An indicator buffer to for drawing the distribution
```



```

double          LabelBuffer[];
//--- A counter of ticks
double          ticks_counter;
//+-----+
//| Custom indicator initialization function          |
//+-----+
void OnInit()
{
//--- Binding an array and an indicator buffer
    SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- turn the indicator buffer around from the present to the past
    ArraySetAsSeries(LabelBuffer,true);
//--- Initialize the generator of random numbers
    MathSrand(GetTickCount());
//--- Initialize the counter of ticks
    ticks_counter=0;
}
//+-----+
//| Custom indicator iteration function              |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- For a zero counter reset the indicator buffer
    if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- Increase the counter
    ticks_counter++;
//--- We should periodically reset the counter ticks, to revive the distribution
    if(ticks_counter>100)
    {
        Print("We've reset the indicator values, let's start filling the cells once again");
        ticks_counter=0;
    }
//--- Get a sample of random values as the sum of three numbers from 0 to 7
    for(int i=0;i<sample_number;i++)
    {
        //--- Calculation of the index of the cell, where the random number falls as the
        int rand_index=0;
        //--- Get three random numbers from 0 to 7
        for(int k=0;k<3;k++)
        {

```



```
    //--- A remainder in the division by 7 will return a value from 0 to 6
    rand_index+=MathRand()%7;
}
//--- Increase the value in the cell number rand_index by 1
LabelBuffer[rand_index]++;
}
//--- Exit the OnCalculate() handler
return(rates_total);
}
```


MathTan

The function returns a tangent of a number.

```
double MathTan(  
    double rad    // argument in radians  
);
```

Parameters

rad

[in] Angle in radians.

Return Value

Tangent of rad. If rad is greater than or equal to 263, or less than or equal to -263, a loss of significance in the result occurs, in which case the function returns an indefinite number.

Note

Instead of MathTan() you can use [tan\(\)](#).

See also

[Real types \(double, float\)](#)

MathIsValidNumber

It checks the correctness of a real number.

```
bool MathIsValidNumber(  
    double number    // number to check  
);
```

Parameters

number

[in] Checked numeric value.

Return Value

It returns true, if the checked value is an acceptable real number. If the checked value is a plus or minus infinity, or "not a number" (NaN), the function returns false.

Example:

```
double abnormal=MathArcsin(2.0);  
if(!MathIsValidNumber(abnormal)) Print("Attention! MathArcsin(2.0) = ",abnormal);
```

See also

[Real types \(double, float\)](#)

MathExp1

Returns the value of the expression $\text{MathExp}(x)-1$.

```
double MathExp1(  
    double value    // power for the number e  
);
```

Parameters

value

[in] The number specifying the power.

Return Value

A value of the double type. In case of overflow the function returns INF (infinity), in case of underflow MathExp returns 0.

Note

At values of x close to 0, the MathExp1(x) function generates much more accurate values than the MathExp(x)-1 function.

Instead of the MathExp1() function you can use the [expm1\(\)](#) function.

See also

[Real types \(double, float\)](#)

MathLog1p

Returns the value of the expression $\text{MathLog}(1+x)$.

```
double MathLog1p(  
    double value    // value to take the logarithm  
);
```

Parameters

value

[in] The value, the logarithm of which is to be calculated.

Return Value

The natural logarithm of the value ($\text{value} + 1$) if successful. If $\text{value} < -1$, the function returns NaN (undefined value). If value is equal to -1 , the function returns INF (infinity).

Note

At values of x close to 0, the $\text{MathLog1p}(x)$ function generates much more accurate values than the $\text{MathLog}(1+x)$ function.

Instead of the $\text{MathLog1p}()$ function you can use the [log1p\(\)](#) function.

See also

[Real types \(double, float\)](#)

MathArccosh

Returns the hyperbolic arccosine.

```
double MathArccosh(  
    double value    // 1 <= value < ∞  
) ;
```

Parameters

value

[in] The value, the hyperbolic arccosine of which is to be calculated.

Return Value

The hyperbolic arccosine of the number. If value is less than +1, the function returns NaN (undefined value).

Note

Instead of the MathArccosh() function you can use the [acosh\(\)](#) function.

See also

[Real types \(double, float\)](#)

MathArcsinh

Returns the hyperbolic arcsine.

```
double MathArcsinh(  
    double value    //  $-\infty < \text{value} < +\infty$   
);
```

Parameters

val

[in] The value, the hyperbolic arcsine of which is to be calculated.

Return Value

The hyperbolic arcsine of the number.

Note

Instead of the MathArcsinh() function you can use the [asinh\(\)](#) function.

See also

[Real types \(double, float\)](#)

MathArctanh

Returns the hyperbolic arctangent.

```
double MathArctanh(  
    double value    // value in the range of -1 < value < 1  
) ;
```

Parameters

value

[in] Number within the range of $-1 < \text{value} < 1$, which represents the tangent.

Return Value

The hyperbolic arctangent of the number.

Note

Instead of the MathArctanh() function you can use the [atanh\(\)](#) function.

MathCosh

Returns the hyperbolic cosine of the number.

```
double MathCosh(  
    double value    // number  
);
```

Parameters

value
[in] Value.

Return Value

The hyperbolic cosine of the number, value within the range of +1 to positive infinity.

Note

Instead of the MathCosh() function you can use the [cosh\(\)](#) function.

MathSinh

Returns the hyperbolic sine of the number.

```
double MathSinh(  
    double value    // number  
);
```

Parameters

value

[in] Value.

Return Value

The hyperbolic sine of the number.

Note

Instead of the MathSinh() function you can use the [sinh\(\)](#) function.

MathTanh

Returns the hyperbolic tangent of the number.

```
double MathTanh(  
    double value    // number  
);
```

Parameters

value
[in] Value.

Return Value

The hyperbolic tangent of the number, value within the range of -1 to +1.

Note

Instead of the MathTanh() function you can use the [tanh\(\)](#) function.

See also

[Real types \(double, float\)](#)

String Functions

This is a group of functions intended for working with data of the [string](#) type.

Function	Action
StringAdd	Adds a string to the end of another string
StringBufferLen	Returns the size of buffer allocated for the string
StringCompare	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1 (minus 1) - if the first string is less than the second one
StringConcatenate	Forms a string of parameters passed
StringFill	Fills out a specified string by selected symbols
StringFind	Search for a substring in a string
StringGetCharacter	Returns the value of a number located in the specified string position
StringInit	Initializes string by specified symbols and provides the specified string length
StringLen	Returns the number of symbols in a string
StringReplace	Replaces all the found substrings of a string by a set sequence of symbols
StringSetCharacter	Returns a copy of a string with a changed value of a symbol in a specified position
StringSplit	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained
StringSubstr	Extracts a substring from a text string starting from a specified position
StringToLower	Transforms all symbols of a selected string to lowercase
StringToUpper	Transforms all symbols of a selected string into capitals
StringTrimLeft	Cuts line feed characters, spaces and tabs in the left part of the string
StringTrimRight	Cuts line feed characters, spaces and tabs in the right part of the string

StringAdd

The function adds a substring to the end of a string.

```
bool StringAdd(
    string& string_var,      // string, to which we add
    string add_substring     // string, which is added
);
```

Parameters

string_var

[in][out] String, to which another one is added.

add_substring

[in] String that is added to the end of a source string.

Return Value

In case of success returns true, otherwise false. In order to get an [error code](#), the [GetLastError\(\)](#) function should be called.

Example:

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

    //--- second method
    start=GetTickCount();
    for(i=0;i<length;i++)
    {
        StringAdd(a,b);
    }
    stop=GetTickCount();
    Print("time for 'StringAdd(a,b)' = ",(stop-start)," milliseconds, i = ",i);

    //--- third method
    start=GetTickCount();
    a="a"; // re-initialize variable a
    for(i=0;i<length;i++)
```



```
{
    StringConcatenate(c,a,b);
}
stop=GetTickCount();
Print("time for 'StringConcatenate(c,a,b)' = ",(stop-start)," milliseconds, i = ",i);
}
```

See also[StringConcatenate](#)

StringBufferLen

The function returns the size of buffer allocated for the string.

```
int StringBufferLen(  
    string string_var    // string  
)
```

Parameters

string_var
[in] String.

Return Value

The value 0 means that the string is constant and buffer size can't be changed. -1 means that the string belongs to the client terminal, and modification of the buffer contents can have indeterminate results.

Note

Minimal buffer size is equal to 16.

Example:

```
void OnStart()  
{  
    long length=1000;  
    string a="a",b="b";  
    //---  
    long i;  
    Print("before: StringBufferLen(a) = ",StringBufferLen(a),  
          "   StringLen(a) = ",StringLen(a));  
    for(i=0;i<length;i++)  
    {  
        StringAdd(a,b);  
    }  
    Print("after: StringBufferLen(a) = ",StringBufferLen(a),  
          "   StringLen(a) = ",StringLen(a));  
}
```

See also

[StringAdd](#), [StringInit](#), [StringLen](#), [StringFill](#)

StringCompare

The function compares two strings and returns the comparison result in form of an integer.

```
int StringCompare(
    const string& string1,           // the first string in the comparison
    const string& string2,           // the second string in the comparison
    bool case_sensitive=true         // case sensitivity mode selection for the
);
```

Parameters

string1

[in] The first string.

string2

[in] The second string.

case_sensitive=true

[in] Case sensitivity mode selection. If it is true, then "A">"a". If it is false, then "A"="a". By default the value is equal to true.

Return Value

- -1 (minus one), if string1<string2
- 0 (zero), if string1=string2
- 1 (one), if string1>string2

Note

The strings are compared symbol by symbol, the symbols are compared in the alphabetic order in accordance with the current code page.

Example:

```
void OnStart()
{
    //--- what is larger - apple or home?
    string s1="Apple";
    string s2="home";

    //--- compare case sensitive
    int result1=StringCompare(s1,s2);
    if(result1>0) PrintFormat("Case sensitive comparison: %s > %s",s1,s2);
    else
    {
        if(result1<0) PrintFormat("Case sensitive comparison: %s < %s",s1,s2);
        else PrintFormat("Case sensitive comparison: %s = %s",s1,s2);
    }

    //--- compare case-insensitive
    int result2=StringCompare(s1,s2,false);
    if(result2>0) PrintFormat("Case insensitive comparison: %s > %s",s1,s2);
    else
```



```
{
    if(result2<0)PrintFormat("Case insensitive comparison: %s < %s",s1,s2);
    else PrintFormat("Case insensitive comparison: %s = %s",s1,s2);
}
/* Result:
    Case-sensitive comparison: Apple < home
    Case insensitive comparison: Apple < home
*/
}
```

See also

[String Type](#), [CharToString\(\)](#), [ShortToString\(\)](#), [StringToCharArray\(\)](#), [StringToShortArray\(\)](#), [StringGetCharacter\(\)](#), [Use of a Codepage](#)

StringConcatenate

The function forms a string of passed parameters and returns the size of the formed string. Parameters can be of any type. Number of parameters can't be less than 2 or more than 64.

```
int StringConcatenate(  
    string& string_var,    // string to form  
    void argument1        // first parameter of any simple type  
    void argument2        // second parameter of any simple type  
    ...                   // next parameter of any simple type  
);
```

Parameters

string_var

[out] String that will be formed as a result of concatenation.

argumentN

[in] Any comma separated values. From 2 to 63 parameters of any simple type.

Return Value

Returns the string length, formed by concatenation of parameters transformed into string type. Parameters are transformed into strings according to the same rules as in [Print\(\)](#) and [Comment\(\)](#).

See also

[StringAdd](#)

StringFill

It fills out a selected string by specified symbols.

```
bool StringFill(  
    string&    string_var,      // string to fill  
    ushort    character        // symbol that will fill the string  
);
```

Parameters

string_var

[in][out] String, that will be filled out by the selected symbol.

character

[in] Symbol, by which the string will be filled out.

Return Value

In case of success returns true, otherwise - false. To get the [error code](#) call [GetLastError\(\)](#).

Note

Filling out a string at place means that symbols are inserted directly to the string without transitional operations of new string creation or copying. This allows to save the operation time.

Example:

```
void OnStart()  
{  
    string str;  
    StringInit(str,20,'_');  
    Print("str = ",str);  
    StringFill(str,0);  
    Print("str = ",str," : StringBufferLen(str) = ", StringBufferLen(str));  
}  
  
// Result  
//   str = _____  
//   str =   : StringBufferLen(str) = 20  
//
```

See also

[StringBufferLen](#), [StringLen](#), [StringInit](#)

StringFind

Search for a substring in a string.

```
int StringFind(  
    string string_value,      // string in which search is made  
    string match_substring,   // what is searched  
    int start_pos=0          // from what position search starts  
);
```

Parameters

string_value

[in] String, in which search is made.

match_substring

[in] Searched substring.

start_pos=0

[in] Position in the string from which search is started.

Return Value

Returns position number in a string, from which the searched substring starts, or -1, if the substring is not found.

StringGetCharacter

Returns value of a symbol, located in the specified position of a string.

```
ushort StringGetCharacter(  
    string string_value,    // string  
    int pos                // symbol position in the string  
);
```

Parameters

string_value

[in] String.

pos

[in] Position of a symbol in the string. Can be from 0 to [StringLen](#)(text) -1.

Return Value

Symbol code or 0 in case of an error. To get the [error code](#) call [GetLastError\(\)](#).

StringInit

Initializes a string by specified symbols and provides the specified string size.

```
bool StringInit(
    string&    string_var,      // string to initialize
    int        new_len=0,       // required string length after initialization
    ushort     character=0      // symbol, by which the string will be filled
);
```

Parameters

string_var

[in][out] String that should be initialized and deinitialized.

new_len=0

[in] String length after initialization. If length=0, it deinitializes the string, i.e. the string buffer is cleared and the buffer address is zeroed.

character=0

[in] Symbol to fill the string.

Return Value

In case of success returns true, otherwise - false. To get the [error code](#) call [GetLastError\(\)](#).

Note

If *character=0* and the length *new_len>0*, the buffer of the string of indicated length will be distributed and filled by zeroes. The string length will be equal to zero, because the whole buffer is filled out by string terminators.

Example:

```
void OnStart()
{
    //---
    string str;
    StringInit(str,200,0);
    Print("str = ",str,": StringBufferLen(str) = ",
        StringBufferLen(str)," StringLen(str) = ",StringLen(str));
}
/* Result
str = : StringBufferLen(str) = 200   StringLen(str) = 0
*/
```

See also

[StringBufferLen](#), [StringLen](#)

StringLen

Returns the number of symbols in a string.

```
int StringLen(  
    string string_value    // string  
);
```

Parameters

string_value

[in] String to calculate length.

Return Value

Number of symbols in a string without the ending zero.

StringReplace

It replaces all the found substrings of a string by a set sequence of symbols.

```
int StringReplace(  
    string&      str,           // the string in which substrings will be replaced  
    const string find,         // the searched substring  
    const string replacement    // the substring that will be inserted to the found  
);
```

Parameters

str

[in][out] The string in which you are going to replace substrings.

find

[in] The desired substring to replace.

replacement

[in] The string that will be inserted instead of the found one.

Return Value

The function returns the number of replacements in case of success, otherwise -1. To get an [error](#) code call the [GetLastError\(\)](#) function.

Note

If the function has run successfully but no replacements have been made (the substring to replace was not found), it returns 0.

The error can result from incorrect *str* or *find* parameters (empty or non-initialized string, see [StringInit\(\)](#)). Besides, the error occurs if there is not enough memory to complete the replacement.

Example:

```
string text="The quick brown fox jumped over the lazy dog.";
int replaced=StringReplace(text,"quick","slow");
replaced+=StringReplace(text,"brown","black");
replaced+=StringReplace(text,"fox","bear");
Print("Replaced: ", replaced,". Result=",text);

// Result
// Replaced: 3. Result=The slow black bear jumped over the lazy dog.
//
```

See also

[StringSetCharacter\(\)](#), [StringSubstr\(\)](#)

StringSetCharacter

Returns copy of a string with a changed character in a specified position.

```
bool StringSetCharacter(
    string&    string_var,      // string
    int        pos,             // position
    ushort     character        // character
);
```

Parameters

string_var

[in][out] String.

pos

[in] Position of a character in a string. Can be from 0 to [StringLen](#)(text).

character

[in] Symbol code Unicode.

Note

If *pos* is less than [string length](#) and the symbol code value = 0, the string is cut off (but the [buffer size](#), distributed for the string remains unchanged). The string length becomes equal to *pos*.

If *pos* is equal to string length, the specified symbol is added at the string end, and the length is enlarged by one.

Example:

```
void OnStart()
{
    string str="0123456789";
    Print("before: str = ",str," ,StringBufferLen(str) = ",
        StringBufferLen(str),"   StringLen(str) = ",StringLen(str));
    //--- add zero value in the middle
    StringSetCharacter(str,6,0);
    Print("after: str = ",str," ,StringBufferLen(str) = ",
        StringBufferLen(str),"   StringLen(str) = ",StringLen(str));
    //--- add symbol at the end
    int size=StringLen(str);
    StringSetCharacter(str,size,'+');
    Print("addition: str = ",str," ,StringBufferLen(str) = ",
        StringBufferLen(str),"   StringLen(str) = ",StringLen(str));
}

/* Result
before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 10
after: str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
*/
```


See also

[StringBufferLen](#), [StringLen](#), [StringFill](#), [StringInit](#)

StringSplit

Gets substrings by a specified separator from the specified string, returns the number of substrings obtained.

```
int StringSplit(
    const string   string_value,      // A string to search in
    const ushort   separator,         // A separator using which substrings will be se
    string         & result[]        // An array passed by reference to get the found
);
```

Parameters

string_value

[in] The string from which you need to get substrings. The string will not change.

pos

[in] The code of the separator character. To get the code, you can use the [StringGetCharacter\(\)](#) function.

result[]

[out] An array of strings where the obtained substrings are located.

Return Value

The number of substrings in the `result[]` array. If the separator is not found in the passed string, only one source string will be placed in the array.

If `string_value` is empty or `NULL`, the function will return zero. In case of an error the function returns -1. To get the [error](#) code, call the [GetLastError\(\)](#) function.

Example:

```
string to_split="_life_is_good_"; // A string to split into substrings
string sep="_";                  // A separator as a character
ushort u_sep;                    // The code of the separator character
string result[];                 // An array to get strings
//--- Get the separator code
u_sep=StringGetCharacter(sep,0);
//--- Split the string to substrings
int k=StringSplit(to_split,u_sep,result);
//--- Show a comment
PrintFormat("Strings obtained: %d. Used separator '%s' with the code %d",k,sep,u_sep);
//--- Now output all obtained strings
if(k>0)
{
    for(int i=0;i<k;i++)
    {
        PrintFormat("result[%d]=" "%s\"",i,result[i]);
    }
}
```


See also

[StringReplace\(\)](#), [StringSubstr\(\)](#), [StringConcatenate\(\)](#)

StringSubstr

Extracts a substring from a text string starting from the specified position.

```
string StringSubstr(  
    string  string_value,      // string  
    int     start_pos,        // position to start with  
    int     length=-1         // length of extracted string  
);
```

Parameters

string_value

[in] String to extract a substring from.

start_pos

[in] Initial position of a substring. Can be from 0 to [StringLen](#)(text) -1.

length=-1

[in] Length of an extracted substring. If the parameter value is equal to -1 or parameter isn't set, the substring will be extracted from the indicated position till the string end.

Return Value

Copy of a extracted substring, if possible. Otherwise returns an empty string.

StringToLower

Transforms all symbols of a selected string into lowercase.

```
bool StringToLower(  
    string& string_var    // string to process  
);
```

Parameters

string_var
[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the [error code](#) call [GetLastError\(\)](#).

StringToUpper

Transforms all symbols of a selected string into capitals.

```
bool StringToUpper(  
    string& string_var    // string to process  
);
```

Parameters

string_var
[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the [error code](#) call [GetLastError\(\)](#).

StringTrimLeft

The function cuts line feed characters, spaces and tabs in the left part of the string till the first meaningful symbol. The string is modified at place.

```
int StringTrimLeft(  
    string& string_var    // string to cut  
);
```

Parameters

string_var

[in][out] String that will be cut from the left.

Return Value

Returns the number of cut symbols.

StringTrimRight

The function cuts line feed characters, spaces and tabs in the right part of the string after the last meaningful symbol. The string is modified at place.

```
int StringTrimRight(  
    string& string_var    // string to cut  
);  
);
```

Parameters

string_var

[in][out] String that will be cut from the right.

Return Value

Returns the number of cut symbols.

Date and Time

This is the group of functions for working with data of [datetime](#) type (an integer that represents the number of seconds elapsed from 0 hours of January 1, 1970).

To arrange high-resolution counters and timers, use the [GetTickCount\(\)](#) function, which produces values in milliseconds.

Function	Action
TimeCurrent	Returns the last known server time (time of the last quote receipt) in the datetime format
TimeTradeServer	Returns the current calculation time of the trade server
TimeLocal	Returns the local computer time in datetime format
TimeGMT	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running
TimeDaylightSavings	Returns the sign of Daylight Saving Time switch
TimeGMTOffset	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch
TimeToStruct	Converts a datetime value into a variable of MqlDateTime structure type
StructToTime	Converts a variable of MqlDateTime structure type into a datetime value

TimeCurrent

Returns the last known server time, time of the last quote receipt for one of the symbols selected in the "Market Watch" window. In the [OnTick\(\)](#) handler, this function returns the time of the received handled tick. In other cases (for example, call in [handlers](#) OnInit(), OnDeinit(), OnTimer() and so on) this is the [time of the last quote receipt](#) for any symbol available in the "Market Watch" window, the time shown in the title of this window. The time value is formed on a trade server and does not depend on the time settings on your computer. There are 2 variants of the function.

Call without parameters

```
datetime TimeCurrent();
```

Call with MqlDateTime type parameter

```
datetime TimeCurrent(  
    MqlDateTime& dt_struct    // structure type variable  
);
```

Parameters

dt_struct

[out] [MqlDateTime](#) structure type variable.

Return Value

Value of [datetime](#) type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the [GetTickCount\(\)](#) function, which produces values in milliseconds.

During testing in the strategy tester, TimeCurrent() is simulated according to historical data.

TimeTradeServer

Returns the calculated current time of the trade server. Unlike [TimeCurrent\(\)](#), the calculation of the time value is performed in the client terminal and depends on the time settings on your computer. There are 2 variants of the function.

Call without parameters

```
datetime TimeTradeServer();
```

Call with MqlDateTime type parameter

```
datetime TimeTradeServer(  
    MqlDateTime& dt_struct    // Variable of structure type  
);
```

Parameters

dt_struct

[out] Variable of structure type [MqlDateTime](#).

Return Value

Value of [datetime](#) type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the [GetTickCount\(\)](#) function, which produces values in milliseconds.

During testing in the strategy tester, TimeTradeServer() is simulated according to historical data and always equal to [TimeCurrent\(\)](#).

TimeLocal

Returns the local time of a computer, where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime TimeLocal();
```

Call with MqlDateTime type parameter

```
datetime TimeLocal(  
    MqlDateTime& dt_struct    // Variable of structure type  
);
```

Parameters

dt_struct

[out] Variable of structure type [MqlDateTime](#).

Return Value

Value of [datetime](#) type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the [GetTickCount\(\)](#) function, which produces values in milliseconds.

During testing in the strategy tester, TimeLocal() is always equal to [TimeCurrent\(\)](#) simulated server time.

TimeGMT

Returns the GMT, which is calculated taking into account the DST switch by the local time on the computer where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime TimeGMT();
```

Call with MqlDateTime type parameter

```
datetime TimeGMT(  
    MqlDateTime& dt_struct    // Variable of structure type  
);
```

Parameters

dt_struct

[out] Variable of structure type [MqlDateTime](#).

Return Value

Value of [datetime](#) type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the [GetTickCount\(\)](#) function, which produces values in milliseconds.

During testing in the strategy tester, TimeGMT() is always equal to [TimeTradeServer\(\)](#) simulated server time.

TimeDaylightSavings

Returns correction for daylight saving time in seconds, if the switch to summer time has been made. It depends on the time settings of your computer.

```
int TimeDaylightSavings();
```

Return Value

If switch to winter (standard) time has been made, it returns 0.

TimeGMTOffset

Returns the current difference between GMT time and the local computer time in seconds, taking into account switch to winter or summer time. Depends on the time settings of your computer.

```
int TimeGMTOffset();
```

Return Value

The value of int type, representing the current difference between [GMT time](#) and the local time of the computer [TimeLocal](#) in seconds.

```
TimeGMTOffset() = TimeGMT() - TimeLocal()
```


TimeToStruct

Converts a value of datetime type (number of seconds since 01.01.1970) into a structure variable [MqlDateTime](#).

```
void TimeToStruct (
    datetime      dt,           // date and time
    MqlDateTime&  dt_struct     // structure for the adoption of values
);
```

Parameters

dt

[in] Date value to convert.

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

No return value.

StructToTime

Converts a structure variable [MqlDateTime](#) into a value of [datetime](#) type and returns the resulting value.

```
datetime StructToTime(  
    MqlDateTime$ dt_struct    // structure of the date and time  
);
```

Parameters

dt_struct

[in] Variable of structure type MqlDateTime.

Return Value

The value of datetime type containing the number of seconds since 01.01.1970.

Account Information

Functions that return parameters of the current account.

Function	Action
<u>AccountInfoDouble</u>	Returns a value of double type of the corresponding account property
<u>AccountInfoInteger</u>	Returns a value of integer type (bool, int or long) of the corresponding account property
<u>AccountInfoString</u>	Returns a value string type corresponding account property

AccountInfoDouble

Returns the value of the corresponding account property.

```
double AccountInfoDouble(  
    int property_id    // identifier of the property  
);
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_DOUBLE](#).

Return Value

Value of [double](#) type.

Example:

```
void OnStart()  
{  
    //--- show all the information available from the function AccountInfoDouble()  
    printf("ACCOUNT_BALANCE = %G",AccountInfoDouble(ACCOUNT_BALANCE));  
    printf("ACCOUNT_CREDIT = %G",AccountInfoDouble(ACCOUNT_CREDIT));  
    printf("ACCOUNT_PROFIT = %G",AccountInfoDouble(ACCOUNT_PROFIT));  
    printf("ACCOUNT_EQUITY = %G",AccountInfoDouble(ACCOUNT_EQUITY));  
    printf("ACCOUNT_MARGIN = %G",AccountInfoDouble(ACCOUNT_MARGIN));  
    printf("ACCOUNT_FREEMARGIN = %G",AccountInfoDouble(ACCOUNT_FREEMARGIN));  
    printf("ACCOUNT_MARGIN_LEVEL = %G",AccountInfoDouble(ACCOUNT_MARGIN_LEVEL));  
    printf("ACCOUNT_MARGIN_SO_CALL = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL));  
    printf("ACCOUNT_MARGIN_SO_SO = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_SO));  
}
```

See also

[SymbolInfoDouble](#), [SymbolInfoString](#), [SymbolInfoInteger](#), [PrintFormat](#)

AccountInfoInteger

Returns the value of the properties of the account.

```
long AccountInfoInteger(
    int property_id    // Identifier of the property
);
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_INTEGER](#).

Return Value

Value of [long](#) type.

Note

The property must be one of the [bool](#), [int](#) or [long](#) types.

Example:

```
void OnStart()
{
    //--- show all the information available from the function AccountInfoInteger()
    printf("ACCOUNT_LOGIN = %d",AccountInfoInteger(ACCOUNT_LOGIN));
    printf("ACCOUNT_LEVERAGE = %d",AccountInfoInteger(ACCOUNT_LEVERAGE));
    bool thisAccountTradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_ALLOWED);
    bool EATradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_EXPERT);
    ENUM_ACCOUNT_TRADE_MODE tradeMode=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE);
    ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_STOPOUT_MODE);

    //--- Inform about the possibility to perform a trade operation
    if(thisAccountTradeAllowed)
        Print("Trade for this account is permitted");
    else
        Print("Trade for this account is prohibited!");

    //--- Find out if it is possible to trade on this account by Expert Advisors
    if(EATradeAllowed)
        Print("Trade by Expert Advisors is permitted for this account");
    else
        Print("Trade by Expert Advisors is prohibited for this account!");

    //--- Find out the account type
    switch(tradeMode)
    {
        case(ACCOUNT_TRADE_MODE_DEMO):
            Print("This is a demo account");
            break;
```



```
case (ACCOUNT_TRADE_MODE_CONTEST):
    Print("This is a competition account");
    break;
default: Print("This is a real account!");
}

/-- Find out the StopOut level setting mode
switch (stopOutMode)
{
    case (ACCOUNT_STOPOUT_MODE_PERCENT):
        Print("The StopOut level is specified percentage");
        break;
    default: Print("The StopOut level is specified in monetary terms");
}
}
```

See also

[Account Information](#)

AccountInfoString

Returns the value of the corresponding account property.

```
string AccountInfoString(  
    int property_id    // Identifier of the property  
);
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_STRING](#).

Return Value

Value of [string](#) type.

Example:

```
void OnStart()  
{  
    //--- Show all the information available from the function AccountInfoString()  
    Print("The name of the broker = ",AccountInfoString(ACCOUNT_COMPANY));  
    Print("Deposit currency = ",AccountInfoString(ACCOUNT_CURRENCY));  
    Print("Client name = ",AccountInfoString(ACCOUNT_NAME));  
    Print("The name of the trade server = ",AccountInfoString(ACCOUNT_SERVER));  
}
```

See also

[Account Information](#)

State Checking

Functions that return parameters of the current state of the client terminal

Function	Action
<u>GetLastError</u>	Returns the last error
<u>IsStopped</u>	Returns true, if an mql5 program has been commanded to stop its operation
<u>UninitializeReason</u>	Returns the code of the reason for deinitialization
<u>TerminalInfoInteger</u>	Returns an integer value of a corresponding property of the mql5 program environment
<u>TerminalInfoDouble</u>	Returns a double value of a corresponding property of the mql5 program environment
<u>TerminalInfoString</u>	Returns a string value of a corresponding property of the mql5 program environment
<u>MQLInfoInteger</u>	Returns an integer value of a corresponding property of a running mql5 program
<u>MQLInfoString</u>	Returns a string value of a corresponding property of a running mql5 program
<u>Symbol</u>	Returns the name of a symbol of the current chart
<u>Period</u>	Returns the current chart timeframe
<u>Digits</u>	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol
<u>Point</u>	Returns the point size of the current symbol in the quote currency

GetLastError

Returns the contents of the system variable [_LastError](#).

```
int GetLastError();
```

Return Value

Returns the value of the last [error](#) that occurred during the execution of an mql5 program.

Note

After the function call, the contents of `_LastError` are not reset. To reset this variable, you need to call [ResetLastError\(\)](#).

See also

[Trade Server Return Codes](#)

IsStopped

Checks the forced shutdown of an mql5 program.

```
bool IsStopped();
```

Return Value

Returns true, if the [_StopFlag](#) system variable contains a value other than 0. A nonzero value is written into `_StopFlag`, if a mql5 program has been commanded to complete its operation. In this case, you must immediately terminate the program, otherwise the program will be completed forcibly from the outside after 3 seconds.

UninitializeReason

Returns the code of a [reason for deinitialization](#).

```
int UninitializeReason();
```

Return Value

Returns the value of [_UninitReason](#) which is formed before [OnDeinit\(\)](#) is called. Value depends on the reasons that led to deinitialization.

TerminalInfoInteger

Returns the value of a corresponding property of the mql5 program environment.

```
int TerminalInfoInteger(  
    int property_id    // identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the [ENUM_TERMINAL_INFO_INTEGER](#) enumeration.

Return Value

Value of int type.

TerminalInfoDouble

Returns the value of a corresponding property of the mql5 program environment.

```
double TerminalInfoDouble(  
    int property_id    // identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the [ENUM_TERMINAL_INFO_DOUBLE](#) enumeration.

Return Value

Value of double type.

TerminalInfoString

Returns the value of a corresponding property of the mql5 program environment. The property must be of string type.

```
string TerminalInfoString(  
    int property_id    // identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the [ENUM_TERMINAL_INFO_STRING](#) enumeration.

Return Value

Value of string type.

MQLInfoInteger

Returns the value of a corresponding property of a running mql5 program.

```
int MQLInfoInteger(  
    int property_id    // identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. Can be one of values of the [ENUM_MQL_INFO_INTEGER](#) enumeration.

Return Value

Value of int type.

MQLInfoString

Returns the value of a corresponding property of a running mql5 program.

```
string MQLInfoString(  
    int property_id    // Identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. Can be one of the [ENUM_MQL_INFO_STRING](#) enumeration.

Return Value

Value of string type.

Symbol

Returns the name of a symbol of the current chart.

```
string Symbol();
```

Return Value

Value of the [_Symbol](#) system variable, which stores the name of the current chart symbol.

Period

Returns the current chart timeframe.

```
ENUM_TIMEFRAMES Period();
```

Return Value

The contents of the [_Period](#) variable that contains the value of the current chart timeframe. The value can be one of the values of the [ENUM_TIMEFRAMES](#) enumeration.

See also

[PeriodSeconds](#), [Chart timeframes](#), [Date and Time](#), [Visibility of objects](#)

Digits

Returns the number of decimal digits determining the accuracy of price of the current chart symbol.

```
int Digits();
```

Return Value

The value of the [_Digits](#) variable which stores the number of decimal digits determining the accuracy of price of the current chart symbol.

Point

Returns the point size of the current symbol in the quote currency.

```
double Point();
```

Return Value

The value of the [_Point](#) variable which stores the point size of the current symbol in the quote currency.

Getting Market Information

These are functions intended for receiving information about the market state.

Function	Action
<u>SymbolsTotal</u>	Returns the number of available (selected in Market Watch or all) symbols
<u>SymbolName</u>	Returns the name of a specified symbol
<u>SymbolSelect</u>	Selects a symbol in the Market Watch window or removes a symbol from the window
<u>SymbolsSynchronized</u>	Checks whether data of a selected symbol in the terminal are <u>synchronized</u> with data on the trade server
<u>SymbolInfoDouble</u>	Returns the double value of the symbol for the corresponding property
<u>SymbolInfoInteger</u>	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property
<u>SymbolInfoString</u>	Returns a value of the string type of a specified symbol for the corresponding property
<u>SymbolInfoMarginRate</u>	Returns the margin rates depending on the order type and direction
<u>SymbolInfoTick</u>	Returns the current prices for the specified symbol in a variable of the <u>MqlTick</u> type
<u>SymbolInfoSessionQuote</u>	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.
<u>SymbolInfoSessionTrade</u>	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.
<u>MarketBookAdd</u>	Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes
<u>MarketBookRelease</u>	Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes
<u>MarketBookGet</u>	Returns a structure array <u>MqlBookInfo</u> containing records of the Depth of Market of a specified symbol

SymbolsTotal

Returns the number of available (selected in Market Watch or all) symbols.

```
int SymbolsTotal(  
    bool selected // True - only symbols in MarketWatch  
);
```

Parameters

selected

[in] Request mode. Can be true or false.

Return Value

If the 'selected' parameter is true, the function returns the number of symbols selected in MarketWatch. If the value is false, it returns the total number of all symbols.

SymbolName

Returns the name of a symbol.

```
string SymbolName(  
    int   pos,           // number in the list  
    bool  selected       // true - only symbols in MarketWatch  
);
```

Parameters

pos

[in] Order number of a symbol.

selected

[in] Request mode. If the value is true, the symbol is taken from the list of symbols selected in MarketWatch. If the value is false, the symbol is taken from the general list.

Return Value

Value of string type with the symbol name.

SymbolSelect

Selects a symbol in the Market Watch window or removes a symbol from the window.

```
bool SymbolSelect(  
    string name,          // symbol name  
    bool select           // add or remove  
);
```

Parameters

name

[in] Symbol name.

select

[in] Switch. If the value is false, a symbol should be removed from MarketWatch, otherwise a symbol should be selected in this window. A symbol can't be removed if the symbol chart is open, or there are open positions for this symbol.

Return Value

In case of failure returns false.

SymbolIsSynchronized

The function checks whether data of a selected symbol in the terminal are synchronized with data on the trade server.

```
bool SymbolIsSynchronized(  
    string name,          // symbol name  
);
```

Parameters

name

[in] Symbol name.

Return value

If data are [synchronized](#), returns 'true'; otherwise returns 'false'.

See also

[SymbolInfoInteger](#), [Organizing Data Access](#)

SymbolInfoDouble

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
double SymbolInfoDouble(
    string          name,          // symbol
    ENUM_SYMBOL_INFO_DOUBLE prop_id // identifier of the property
);
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool SymbolInfoDouble(
    string          name,          // symbol
    ENUM_SYMBOL_INFO_DOUBLE prop_id, // identifier of the property
    double&         double_var    // here we accept the property value
);
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the [ENUM_SYMBOL_INFO_DOUBLE](#) enumeration.

double_var

[out] Variable of double type receiving the value of the requested property.

Return Value

The value of double type. In case of execution failure, information about the [error](#) can be obtained using [GetLastError\(\)](#) function:

- 5040 - invalid string parameter for specifying a symbol name,
- 4301 - unknown symbol (financial instrument),
- 4302 - symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 - invalid identifier of a symbol property.

Note

It is recommended to use [SymbolInfoTick\(\)](#) if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use [SymbolInfoTick\(\)](#) function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:

```
void OnTick()
```



```
{
//--- obtain spread from the symbol properties
bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
string comm=StringFormat("Spread %s = %I64d points\r\n",
                        spreadfloat?"floating":"fixed",
                        SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- now let's calculate the spread by ourselves
double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
double spread=ask-bid;
int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
comm=comm+"Calculated spread = "+(string)spread_points+" points";
Comment(comm);
}
```


SymbolInfoInteger

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
long SymbolInfoInteger(
    string          name,          // symbol
    ENUM_SYMBOL_INFO_INTEGER prop_id // identifier of a property

);
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool SymbolInfoInteger(
    string          name,          // symbol
    ENUM_SYMBOL_INFO_INTEGER prop_id, // identifier of a property
    long&          long_var      // here we accept the property value

);
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the [ENUM_SYMBOL_INFO_INTEGER](#) enumeration.

long_var

[out] Variable of the long type receiving the value of the requested property.

Return Value

The value of long type. In case of execution failure, information about the [error](#) can be obtained using [GetLastError\(\)](#) function:

- 5040 - invalid string parameter for specifying a symbol name,
- 4301 - unknown symbol (financial instrument),
- 4302 - symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 - invalid identifier of a symbol property.

Note

It is recommended to use [SymbolInfoTick\(\)](#) if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use [SymbolInfoTick\(\)](#) function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:


```
void OnTick()
{
    //--- obtain spread from the symbol properties
    bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
    string comm=StringFormat("Spread %s = %I64d points\r\n",
                             spreadfloat?"floating":"fixed",
                             SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
    //--- now let's calculate the spread by ourselves
    double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
    double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    double spread=ask-bid;
    int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
    comm=comm+"Calculated spread = "+(string)spread_points+" points";
    Comment(comm);
}
```


SymbolInfoString

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
string SymbolInfoString(
    string          name,           // Symbol
    ENUM_SYMBOL_INFO_STRING prop_id // Property identifier
);
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a placeholder variable passed by reference in the last parameter.

```
bool SymbolInfoString(
    string          name,           // Symbol
    ENUM_SYMBOL_INFO_STRING prop_id, // Property identifier
    string&         string_var     // here we accept the property value
);
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the [ENUM_SYMBOL_INFO_STRING](#) enumeration.

string_var

[out] Variable of the string type receiving the value of the requested property.

Return Value

The value of string type. In case of execution failure, information about the [error](#) can be obtained using [GetLastError\(\)](#) function:

- 5040 - invalid string parameter for specifying a symbol name,
- 4301 - unknown symbol (financial instrument),
- 4302 - symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 - invalid identifier of a symbol property.

Note

It is recommended to use [SymbolInfoTick\(\)](#) if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use [SymbolInfoTick\(\)](#) function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

SymbolInfoMarginRate

Returns the margin rates depending on the order type and direction.

```
bool SymbolInfoMarginRate(  
    string          name,           // symbol name  
    ENUM_ORDER_TYPE order_type,     // order type  
    double&         initial_margin_rate, // initial margin rate  
    double&         maintenance_margin_rate // maintenance margin rate  
);
```

Parameters

name

[in] Symbol name.

order_type

[in] Order type.

initial_margin_rate

[in] A [double](#) type variable for receiving an initial margin rate. Initial margin is a security deposit for 1 lot deal in the appropriate direction. Multiplying the rate by the initial margin, we receive the amount of funds to be reserved on the account when placing an order of the specified type.

maintenance_margin_rate

[out] A [double](#) type variable for receiving a maintenance margin rate. Maintenance margin is a minimum amount for maintaining an open position of 1 lot in the appropriate direction. Multiplying the rate by the maintenance margin, we receive the amount of funds to be reserved on the account after an order of the specified type is activated.

Return Value

Returns true if request for properties is successful, otherwise false.

SymbolInfoTick

The function returns current prices of a specified symbol in a variable of the MqlTick type.

```
bool SymbolInfoTick(  
    string    symbol,      // symbol name  
    MqlTick& tick          // reference to a structure  
);
```

Parameters

symbol

[in] Symbol name.

tick

[out] Link to the structure of the [MqlTick](#) type, to which the current prices and time of the last price update will be placed.

Return Value

The function returns true if successful, otherwise returns false.

SymbolInfoSessionQuote

Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.

```
bool SymbolInfoSessionQuote(  
    string          name,           // symbol name  
    ENUM_DAY_OF_WEEK day_of_week,   // day of the week  
    uint            session_index,   // session index  
    datetime&       from,           // time of the session beginning  
    datetime&       to              // time of the session end  
);
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration [ENUM_DAY_OF_WEEK](#).

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return Value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

See also

[Symbol Properties](#), [TimeToStruct](#), [Data Structures](#)

SymbolInfoSessionTrade

Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.

```
bool SymbolInfoSessionTrade(  
    string          name,           // symbol name  
    ENUM_DAY_OF_WEEK day_of_week,   // day of the week  
    uint            session_index,   // session index  
    datetime&       from,           // session beginning time  
    datetime&       to              // session end time  
);
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration [ENUM_DAY_OF_WEEK](#).

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

See also

[Symbol Properties](#), [TimeToStruct](#), [Data Structures](#)

MarketBookAdd

Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes.

```
bool MarketBookAdd(  
    string symbol    // symbol  
);
```

Parameters

symbol

[in] The name of a symbol, whose Depth of Market is to be used in the Expert Advisor or script.

Return Value

The true value if opened successfully, otherwise false.

Note

Normally, this function must be called from the [OnInit\(\)](#) function or in the class constructor. To handle incoming alerts, in the Expert Advisor program must contain the function void [OnBookEvent](#)(string& symbol).

See also

[Structure of Depth of Market](#), [Structures and Classes](#)

MarketBookRelease

Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes.

```
bool MarketBookRelease(  
    string symbol    // symbol  
);
```

Parameters

symbol

[in] Symbol name.

Return Value

The true value if closed successfully, otherwise false.

Note

Normally, this function must be called from the [OnDeinit\(\)](#) function, if the corresponding [MarketBookAdd\(\)](#) function has been called in the [OnInit\(\)](#) function. Or it must be called from the class destructor, if the corresponding MarketBookAdd() function has been called from the class constructor.

See also

[Structure of Depth of Market](#), [Structures and Classes](#)

MarketBookGet

Returns a structure array [MqlBookInfo](#) containing records of the Depth of Market of a specified symbol.

```
bool MarketBookGet (
    string      symbol,      // symbol
    MqlBookInfo& book[]      // reference to an array
);
```

Parameters

symbol

[in] Symbol name.

book[]

[in] Reference to an array of Depth of Market records. The array can be pre-allocated for a sufficient number of records. If a [dynamic array](#) hasn't been pre-allocated in the operating memory, the client terminal will distribute the array itself.

Return Value

Returns true in case of success, otherwise false.

Note

The Depth of Market must be pre-opened by the [MarketBookAdd\(\)](#) function.

Example:

```
MqlBookInfo priceArray[];
bool getBook=MarketBookGet(NULL,priceArray);
if(getBook)
{
    int size=ArraySize(priceArray);
    Print("MarketBookInfo for ",Symbol());
    for(int i=0;i<size;i++)
    {
        Print(i+":",priceArray[i].price
            + "    Volume = "+priceArray[i].volume,
            " type = ",priceArray[i].type);
    }
}
else
{
    Print("Could not get contents of the symbol DOM ",Symbol());
}
```

See also

[Structure of Depth of Market](#), [Structures and Classes](#)

Access to Timeseries and Indicator Data

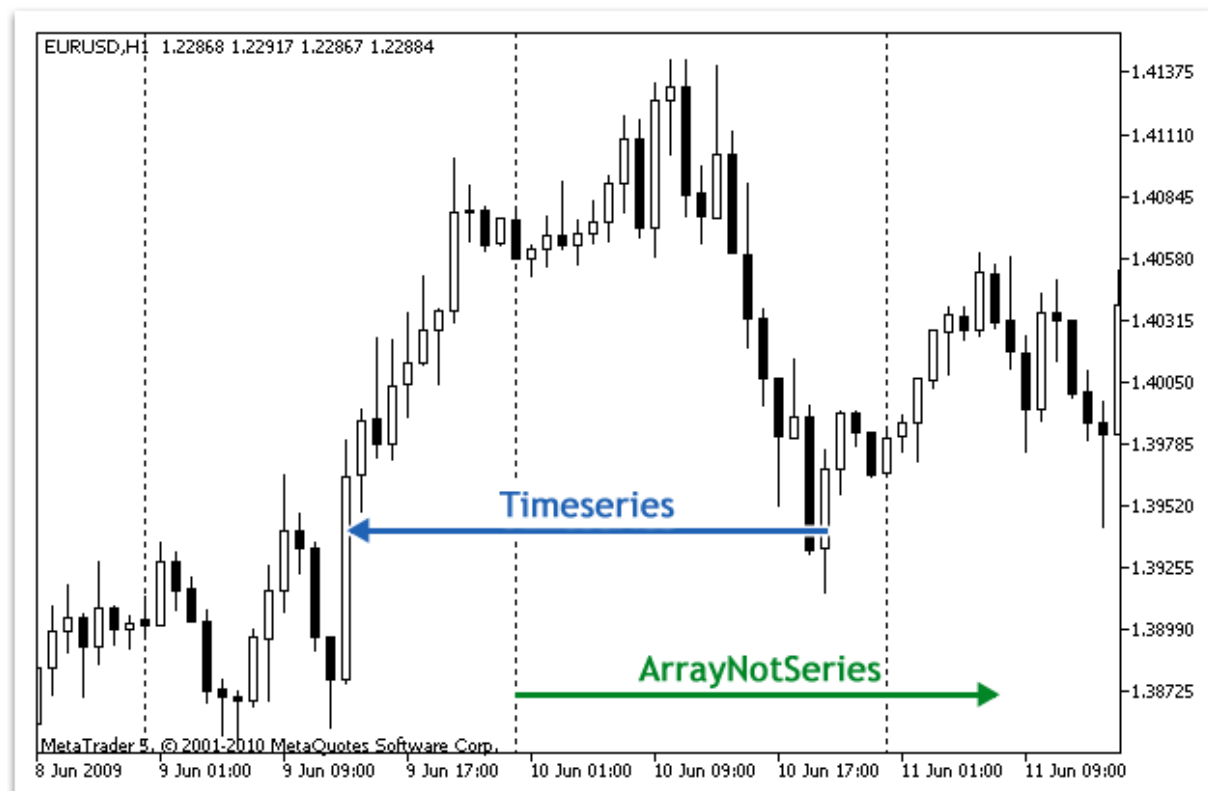
These are functions for working with timeseries and indicators. A timeseries differs from the usual data array by its reverse ordering - elements of timeseries are indexed from the end of an array to its begin (from the most recent data to the oldest ones). To copy the time-series values and indicator data, it's recommended to use [dynamic arrays](#) only, because copying functions are designed to allocate the necessary size of arrays that receive values.

There is an **important exception** to this rule: if timeseries and indicator values need to be copied often, for example at each call of [OnTick\(\)](#) in Expert Advisors or at each call of [OnCalculate\(\)](#) in indicators, in this case one should better use [statically distributed arrays](#), because **operations of memory allocation** for dynamic arrays **require additional time**, and this will have effect during testing and optimization.

When using functions accessing timeseries and indicator values, indexing direction should be taken into account. This is described in the [Indexing Direction in Arrays, Buffers and Timeseries](#) section.

Access to indicator and timeseries data is implemented irrespective of the fact whether the requested data are ready (the so called [asynchronous access](#)). This is critically important for the calculation of custom indicator, so if there are no data, functions of *Copy...()* type immediately return an error. However, when accessing from Expert Advisors and scripts, several attempts to receive data are made in a small pause, which is aimed at providing some time necessary to download required timeseries or to calculate indicator values.

The [Organizing Data Access](#) section describes details of receiving, storing and requesting price data in the MetaTrader 5 client terminal.



It is historically accepted that an access to the price data in an array is performed from the end of the data. Physically, the new data are always written at the array end, but the index of the array is always

equal to zero. The 0 index in the timeseries array denotes data of the current bar, i.e. the bar that corresponds to the unfinished time interval in this timeframe.

A timeframe is the time period, during which a single price bar is formed. There are 21 predefined [standard timeframes](#).

Function	Action
SeriesInfoInteger	Returns information about the state of historical data
Bars	Returns the number of bars count in the history for a specified symbol and period
BarsCalculated	Returns the number of calculated data in an indicator buffer or -1 in the case of error (data hasn't been calculated yet)
IndicatorCreate	Returns the handle to the specified technical indicator created by an array of MqlParam type parameters
IndicatorParameters	Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters
IndicatorRelease	Removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else
CopyBuffer	Gets data of a specified buffer from a specified indicator into an array
CopyRates	Gets history data of the Rates structure for a specified symbol and period into an array
CopyTime	Gets history data on bar opening time for a specified symbol and period into an array
CopyOpen	Gets history data on bar opening price for a specified symbol and period into an array
CopyHigh	Gets history data on maximal bar price for a specified symbol and period into an array
CopyLow	Gets history data on minimal bar price for a specified symbol and period into an array
CopyClose	Gets history data on bar closing price for a specified symbol and period into an array
CopyTickVolume	Gets history data on tick volumes for a specified symbol and period into an array
CopyRealVolume	Gets history data on trade volumes for a specified symbol and period into an array

<u>CopySpread</u>	Gets history data on spreads for a specified symbol and period into an array
<u>CopyTicks</u>	Gets ticks in the MqlTick format into ticks_array
<u>CopyTicksRange</u>	Gets ticks in the MqlTick format within the specified date range to ticks_array

Despite the fact that by using the [ArraySetAsSeries\(\)](#) function it is possible to set up in [arrays](#) access to elements like that in timeseries, it should be remembered that the array elements are physically stored in one and the same order - only indexing direction changes. To demonstrate this fact let's perform an example:

```

datetime TimeAsSeries[];
//--- set access to the array like to a timeseries
ArraySetAsSeries(TimeAsSeries,true);
ResetLastError();
int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
if(copied<=0)
{
    Print("The copy operation of the open time values for last 10 bars has failed");
    return;
}
Print("TimeCurrent =",TimeCurrent());
Print("ArraySize(Time) =",ArraySize(TimeAsSeries));
int size=ArraySize(TimeAsSeries);
for(int i=0;i<size;i++)
{
    Print("TimeAsSeries["+i+"] =",TimeAsSeries[i]);
}

datetime ArrayNotSeries[];
ArraySetAsSeries(ArrayNotSeries,false);
ResetLastError();
copied=CopyTime(NULL,0,0,10,ArrayNotSeries);
if(copied<=0)
{
    Print("The copy operation of the open time values for last 10 bars has failed");
    return;
}
size=ArraySize(ArrayNotSeries);
for(int i=size-1;i>=0;i--)
{
    Print("ArrayNotSeries["+i+"] =",ArrayNotSeries[i]);
}

```

As a result we will get the output like this:

```
TimeCurrent = 2009.06.11 14:16:23
```



```
ArraySize(Time) = 10
TimeAsSeries[0] = 2009.06.11 14:00:00
TimeAsSeries[1] = 2009.06.11 13:00:00
TimeAsSeries[2] = 2009.06.11 12:00:00
TimeAsSeries[3] = 2009.06.11 11:00:00
TimeAsSeries[4] = 2009.06.11 10:00:00
TimeAsSeries[5] = 2009.06.11 09:00:00
TimeAsSeries[6] = 2009.06.11 08:00:00
TimeAsSeries[7] = 2009.06.11 07:00:00
TimeAsSeries[8] = 2009.06.11 06:00:00
TimeAsSeries[9] = 2009.06.11 05:00:00

ArrayNotSeries[9] = 2009.06.11 14:00:00
ArrayNotSeries[8] = 2009.06.11 13:00:00
ArrayNotSeries[7] = 2009.06.11 12:00:00
ArrayNotSeries[6] = 2009.06.11 11:00:00
ArrayNotSeries[5] = 2009.06.11 10:00:00
ArrayNotSeries[4] = 2009.06.11 09:00:00
ArrayNotSeries[3] = 2009.06.11 08:00:00
ArrayNotSeries[2] = 2009.06.11 07:00:00
ArrayNotSeries[1] = 2009.06.11 06:00:00
ArrayNotSeries[0] = 2009.06.11 05:00:00
```

As we see from the output, as the index of TimeAsSeries array increases, the time value of the index decreases, i.e. we move from the present to the past. For the common array ArrayNotSeries the result is different - as index grows, we move from past to present.

See Also

[ArrayIsDynamic](#), [ArrayGetAsSeries](#), [ArraySetAsSeries](#), [ArrayIsSeries](#)

Indexing Direction in Arrays, Buffers and Timeseries

The default indexing of all arrays and indicator buffers is left to right. The index of the first element is always equal to zero. Thus, the very first element of an array or indicator buffer with index 0 is by default on the extreme left position, while the last element is on the extreme right position.

An indicator buffer is a [dynamic array](#) of type double, whose size is managed by the client terminals, so that it always corresponds to the number of bars the indicator is calculated on. A usual dynamic array of type double is assigned as an indicator buffer using the [SetIndexBuffer\(\)](#) function. Indicator buffers do not require setting of their size using function [ArrayResize\(\)](#) - this will be done by the executing system of the terminal.

[Timeseries](#) are arrays with reverse indexing, i.e. the first element of a timeseries is in the extreme right position, and the last element is in the extreme left position. Timeseries being used for storing history price data and contain the time information, we can say that the newest data are placed in the extreme right position of the timeseries, while the oldest data are in the extreme left position.

So the timeseries element with index 0 contains the information about the latest quote of a symbol. If a timeseries contains data on a daily timeframe, data of the current yet uncompleted day are located on the zero position, and the position with index 1 contains yesterday data.

Changing the Indexing Direction

Function [ArraySetAsSeries\(\)](#) allows changing the method of accessing elements of a dynamic array; the physical order of data storing in the computer memory is not changed at that. This function simply changes the method of addressing array elements, so when copying one array to another using function [ArrayCopy\(\)](#), the contents of the recipient array will not depend on the indexing direction in the source array.

Direction of indexing cannot be changed for statically distributed arrays. Even if an array was passed as a parameter to a function, attempts to change the indexing direction inside this function will bring no effect.

For indicator buffers, like for usual arrays, indexing direction can also be set as backward (like in timeseries), i.e. reference to the zero position in the indicator buffer will mean reference to the last value on the corresponding indicator buffer and this will correspond to the value of the indicator on the latest bar. Still, the physical location of indicator bars will be unchanged.

Receiving Price Data in Indicators

Each [custom indicator](#) must necessarily contain the [OnCalculate\(\)](#) function, to which price data required for calculating values in indicator buffers are passed. Indexing direction in these passed arrays can be found out using function [ArrayGetAsSeries\(\)](#).

Arrays [passed to the function](#) reflect price data, i.e. these arrays have the sign of a timeseries and function [ArrayIsSeries\(\)](#) will return true when checking these arrays. However, in any case indexing direction should be checked only by function [ArrayGetAsSeries\(\)](#).

In order not to be dependent on default values, [ArraySetAsSeries\(\)](#) should be unconditionally called for the arrays you are going to work with, and set the required direction.

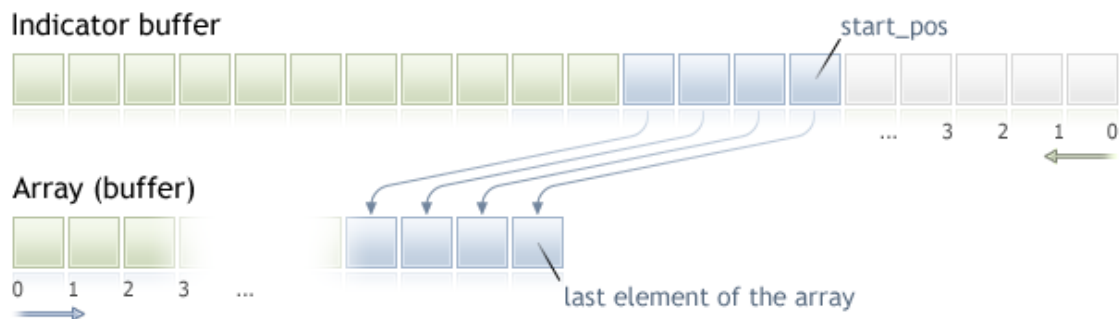
Receiving Price Data and Indicator Values

Default indexing direction of all arrays in Expert Advisors, indicators and scripts is left-to-right. If necessary, in any mql5 program you can request timeseries values on any symbol and timeframe, as well as values of indicators calculated on any symbol and timeframe.

Use functions `Copy...()` for these purposes:

- [CopyBuffer](#) - copy values of an indicator buffer to an array of double type;
- [CopyRates](#) - copy price history to an array of structures [MqlRates](#);
- [CopyTime](#) - copy Time values to an array of datetime type;
- [CopyOpen](#) - copy Open values to an array of double type;
- [CopyHigh](#) - copy High values to an array of double type;
- [CopyLow](#) - copy Low values to an array of double type;
- [CopyClose](#) - copy Close values to an array of double type;
- [CopyTickVolume](#) - copy tick volumes to an array of long type;
- [CopyRealVolume](#) - copy equity volumes to a long type array;
- [CopySpread](#) - copy the spread history to an array of int type;

All these functions work in a similar way. Let's consider the data obtaining mechanism on the example of `CopyBuffer()`. It is implied that the indexing direction of requested data is that of timeseries, and the position with index 0 (zero) stores data of the current yet uncompleted bar. In order to get access to these data we need to copy the necessary volume of data into the recipient array, e.g. into array *buffer*.



When copying we need to specify the starting position in the source array, starting from which data will be copied to the recipient array. In case of success, the specified number of elements will be copied to the recipient array from the source array (from the indicator buffer in this case). Irrespective of the indexing value set in the recipient array, copying is always performed as is shown in the above figure.

If it is expected that price data will be handled in a loop with a large number of iterations, it is advisable that you check the fact of forced program termination using the [IsStopped\(\)](#) function:

```
int copied=CopyBuffer(ma_handle, // Indicator handle
    0, // The index of the indicator buffer
    0, // Start position for copying
    number, // Number of values to copy
    Buffer // The array that receives the values
```



```

        );
    if(copied<0) return;
    int k=0;
    while(k<copied && !IsStopped())
    {
        //-- Get the value for the k index
        double value=Buffer[k];
        // ...
        // work with value
        k++;
    }

```

Example:

```

input int per=10; // period of the exponent
int ma_handle;    // indicator handle
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--
    ma_handle=iMA(_Symbol,0,per,0,MODE_EMA,PRICE_CLOSE);
    //--
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
    //--
    double ema[10];
    int copied=CopyBuffer(ma_handle, // indicator handle
                          0,          // index of the indicator buffer
                          0,          // starting position to copy from
                          10,         // number of values for copying
                          ema         // value receiving array
    );

    if(copied<0) return;
    // .... further code
}

```

See also

[Organizing Data Access](#)

Organizing Data Access

In this section questions connected with obtaining, storing and requesting price data ([timeseries](#)) are considered.

Receiving Data from a Trade Server

Before price data become available in the MetaTrader 5 terminal, they must be received and processed. To receive data, connection to the MetaTrader 5 trade server must be established. Data are received in the form of packed blocks of minute bars from the server upon the request of a terminal.

The mechanism of server reference for data doesn't depend on how the request has been initiated - by a user when navigating in a chart or in a program way in the MQL5 language.

Storing Intermediate Data

Data received from a server are automatically unpacked and saved in the HCC intermediate format. Data on each symbol are written into a separate folder: *terminal_directory\bases\server_name\history\symbol_name*. For example, data on EURUSD received from the MetaQuotes-Demo server will be stored in *terminal_directory\bases\MetaQuotes-Demo\history\EURUSD*.

Data are written into files with .hcc extension. Each file stores data of minute bars for one year. For example, the file named 2009.hcc in the EURUSD folder contains minute bars of EURUSD for year 2009. These files are used for preparing price data for all timeframes and are not intended for direct access.

Obtaining Data on a Necessary Timeframe out of Intermediate Data

Intermediate HCC files are used as the data source for building price data for requested timeframes in the HC format. Data of HC format are timeseries that are maximally prepared for a quick access. They are created upon a request of a chart or a MQL5 program. The volume of data should not exceed the value of the "Max bars in charts" parameter. Data are stored for further using in files with hc extension.

To save resources, data on a timeframe are stored and saved in RAM only if necessary. If not called for a long time, they are released from RAM and saved into a file. For each timeframe, data are prepared regardless of whether there are ready data for other timeframes or not. Rules of forming and accessing data are the same for all timeframes. I.e., despite the fact that the unit data stored in HCC is one minute (M1), the availability of HCC data doesn't mean the availability of data on M1 timeframe as HC in the same volume.

Receipt of new data from a server calls automatic update of used price data in HC format of all timeframes. It also leads to the recalculation of all indicators that implicitly use them as input data for calculations.

Parameter "Max bars in chart"

The "Max bars in charts" parameter restricts number of bars in HC format available to charts, indicators and mql5 programs. This is valid for all available timeframes and serves, first of all, to save computer resources.

When setting a large value of this parameter, it should be remembered, that if deep history price data for small timeframes are available, memory used for storing timeseries and indicator buffers can become hundreds of megabytes and reach the RAM restriction for the client terminal program (2Gb for 32-bit applications of MS Windows).

Change of the "Max bars in charts" comes into effect after the client terminal is restarted. Change of this parameter causes neither automatic referring to a server for additional data, nor forming of additional bars of timeseries. Additional price data are requested from the server, and timeseries are updated taking into account the new limitation, in case of either chart scroll to the area with no data, or when data are requested by a mql5 program.

Volume of data requested from the server corresponds to the required number of bars of this timeframe with the "Max bars in charts" parameter taken into account. The restriction set by this parameter is not strict, and in some cases the number of available bars for a timeframe can be a little more than the current parameter value.

Data Availability

Presence of data on HCC format or even in the prepared for using HC format does not always denote the absolute availability of these data to be shown in a chart or to be used in MQL5 programs.

When accessing to price data or indicator values from a mql5 program it should be remembered that their availability in a certain moment of time or starting from a certain moment of time is not guaranteed. It is connected with the fact that with the purpose of saving resources, the full copy of data necessary for a mql5 program isn't stored in MetaTrader 5; only direct access to the terminal database is given.

The price history for all timeframes is built from common data of HCC format, and any update of data from a server leads to the update of data for all timeframes and to the recalculation of indicators. Due to this access to data can be closed, even if these data were available a moment ago.

Synchronization of the Terminal Data and Server Data

Since a mql5 program can call data from any symbol and timeframe, there is a possibility that data of a necessary timeseries are not formed yet in the terminal or the necessary price data aren't synchronized with the trade server. In this case it's hard to predict the latency time.

Algorithms using "do-nothing" loops are not the best solution. The only exception in this case are scripts, because they do not have any alternative algorithm choice due to not having event handling. For custom indicators such algorithms, as well as any other "do-nothing" loops are strongly not recommended, because they lead to termination of calculation of all indicators and any other handling of price data of the symbol.

For Expert Advisors and indicators, it is better to use the [event model](#) of handling. If during handling of OnTick() or OnCalculate() event, data receipt for the required timeseries failed, you should exit the event handler, relying on the access availability during the next call of the handler.

Example of a Script for Adding History

Let's consider the example of a script that executes a request to receive history for the selected symbol from a trade server. The script is intended for running in a chart of a selected symbol;

timeframe doesn't matter, because, as it was mentioned above, price data are received from a trade server as packed one minute data, from which any predefined timeseries is constructed then.

Write all actions concerning data receipt as a separate function `CheckLoadHistory(symbol, timeframe, start_date)`:

```
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
{
}
```

The `CheckLoadHistory()` function is designed as a universal function that can be called from any program (Expert Advisor, script or indicator); and therefore it requires three input parameters: symbol name, period and start date to indicate the beginning of price history you need.

Insert necessary checks into the function code before requesting the missing history. First of all, we should make sure that the symbol name and period value are correct:

```
if(symbol==NULL || symbol=="") symbol=Symbol();
if(period==PERIOD_CURRENT) period=Period();
```

Then let's make sure that the symbol is available in the MarketWatch window, i.e., the history for the symbol will be available when sending a request to a trade server. If there is no such a symbol in MarketWatch, add it using the [SymbolSelect\(\)](#) function.

```
if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
{
    if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
    SymbolSelect(symbol,true);
}
```

Now we should receive the start date of the available history for the indicated symbol/period pair. Perhaps, the value of the input parameter `startdate`, passed to `CheckLoadHistory()`, is within the available history; then request to a trade server is not needed. In order to obtain the very first date for the symbol-period as of the moment, the [SeriesInfoInteger\(\)](#) function with the [SERIES_FIRSTDATE](#) modifier is used.

```
SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
if(first_date>0 && first_date<=start_date) return(1);
```

The next important check is checking the type of the program, from which the function is called. Note that it is not desirable to send a request to update the timeseries from indicator with the same period. The undesirability of requesting data on the same symbol-period as that of the indicator is conditioned by the fact that update of history data is performed in the same thread where the indicator operates. So the possibility of deadlock occurrence is high. To check this use the [MQL5InfoInteger\(\)](#) function with the [MQL5_PROGRAM_TYPE](#) modifier.

```
if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sym
return(-4);
```

If all the checks have been passed successfully, make the last attempt to do without referring to the trade server. First let's find out the start date, for which minute data in HCC format are available. Request this value using the `SeriesInfoInteger()` function with the [SERIES_TERMINAL_FIRSTDATE](#) modifier and again compare it to the value of the `start_date` parameter.


```

if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
{
    //--- there is loaded data to build timeseries
    if(first_date>0)
    {
        //--- force timeseries build
        CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
        //--- check date
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
    }
}

```

If after all the checks the execution thread is still in the body of the CheckLoadHistory() function, it means there is a necessity to request the missing price data from a trade server. First, return the value of "Max bars in chart" using the [TerminalInfoInteger\(\)](#) function:

```
int maxBars=TerminalInfoInteger(TERMINAL_MAXBARS);
```

We'll need it to prevent requesting extra data. Then find the very first date in the symbol history on the trade server (regardless of the period) using already known function SeriesInfoInteger() with the [SERIES_SERVER_FIRSTDATE](#) modifier.

```

datetime first_server_date=0;
while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date))
    Sleep(5);

```

Since the request is an asynchronous operation, the function is called in the loop with a small delay of 5 milliseconds until the first_server_date variable receives a value, or the loop execution is terminated by a user ([IsStopped\(\)](#) will return *true* in this case). Let's indicate a correct value of the start date, starting from which we request price data from a trade server.

```

if(first_server_date>start_date) start_date=first_server_date;
if(first_date>0 && first_date<first_server_date)
    Print("Warning: first server date ",first_server_date," for ",
symbol," does not match to first series date ",first_date);

```

If the start date *first_server_date* of the server is lower than the start date *first_date* of the symbol in HCC format, the corresponding entry will be output in the journal.

Now we are ready to make a request to a trade server asking for missing price data. Make the request in the form of a loop and start filling out its body:

```

while(!IsStopped())
{
    //1. wait for synchronization between the re-built timeseries and intermediate bars
    //2. receive the current number of bars in this timeseries
    //    if bars is larger than MaxBarsInChart, we can exit, work is over
    //3. obtain the start date first_date in the re-built timeseries and compare it
    //    if first_date is lower than start_date, we can exit, work is over
    //4. request from a server a new part of history - 100 bars starting from last date
}

```


The first three points are implemented by already known means.

```
while(!IsStopped())
{
    //--- 1.wait till timeseries re-build process is over
    while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
        Sleep(5);
    //--- 2.request how many bars we have
    int bars=Bars(symbol,period);
    if(bars>0)
    {
        //--- bars more than ones that can be drawn in the chart, exit
        if(bars>=max_bars) return(-2);
        //--- 3. return the current start date in the timeseries
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            // start date was earlier than that requested, task completed
            if(first_date>0 && first_date<=start_date) return(0);
    }
    //4. Request from a server a new part of history - 100 bars starting from last a
}
```

The last fourth point is left - requesting history. We can't refer to a server directly, but any [Copy-function](#) automatically initiates request sending to a server, if the history in HCC format is not enough. Since the time of the very first start date in the *first_date* variable is the simple and natural criterion to evaluate the request execution degree, then the easiest way is to use the [CopyTime\(\)](#) function.

When calling functions that copy any data from timeseries, it should be noted that the *start* parameter (number of the bar, starting from which price data should be copied) must always be within the available terminal history. If you have only 100 bars, it meaningless to try copying 300 bars starting from the bar with the index 500. Such a request will be understood as an erroneous and won't be handled, i.e. no additional history will be loaded from a trade server.

That's why we'll copy bars in groups of 100 starting from the bar with the *bars* index. This will provide the smooth loading of missing history from a trade server. Actually a little more than the requested 100 bars will be loaded, while server sends oversized history.

```
int copied=CopyTime(symbol,period,bars,100,times);
```

After the copying operation, we should analyze the number of copied elements. If the attempt fails, then value of the *copied* will be equal to null and the value of the *fail_cnt* counter will be increased by 1. After 100 failing attempts, the operation of the function will be stopped.

```
int fail_cnt=0;
...
int copied=CopyTime(symbol,period,bars,100,times);
if(copied>0)
{
    //--- check data
    if(times[0]<=start_date) return(0); // the copied value is smaller, ready
    if(bars+copied>=max_bars) return(-2); // bars are more than can be drawn in the
```



```

        fail_cnt=0;
    }
    else
    {
        //--- no more than 100 failing attempts in succession
        fail_cnt++;
        if(fail_cnt>=100) return(-5);
        Sleep(10);
    }

```

So, not only correct handling of the current situation at each moment of execution is implemented in the function, but also the termination code is returned, that can be handled after calling the CheckLoadHistory() function for getting additional information. For example, this way:

```

int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
switch(res)
{
    case -1 : Print("Unknown symbol ",InpLoadedSymbol); break;
    case -2 : Print("More requested bars than can be drawn in the chart"); break;
    case -3 : Print("Execution stopped by user"); break;
    case -4 : Print("Indicator mustn't load its own data"); break;
    case -5 : Print("Loading failed"); break;
    case 0 : Print("All data loaded"); break;
    case 1 : Print("Already available data in timeseries are enough"); break;
    case 2 : Print("Timeseries is built from available terminal data"); break;
    default : Print("Execution result undefined");
}

```

The full code of the function can be found in the example of a script that shows the correct organization of access to any data with the handling of request's results.

Code:

```

//+-----+
//|                                     TestLoadHistory.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.02"
#property script_show_inputs
//--- input parameters
input string        InpLoadedSymbol="NZDUSD"; // Symbol to be load
input ENUM_TIMEFRAMES InpLoadedPeriod=PERIOD_H1; // Period to be loaded
input datetime      InpStartDate=D'2006.01.01'; // Start date
//+-----+
//| Script program start function |
//+-----+
void OnStart()

```



```

{
    Print("Start load",InpLoadedSymbol+", "+GetPeriodName(InpLoadedPeriod), "from", InpStartDate);
//---
    int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
    switch(res)
    {
        case -1 : Print("Unknown symbol ",InpLoadedSymbol); break;
        case -2 : Print("Requested bars more than max bars in chart"); break;
        case -3 : Print("Program was stopped"); break;
        case -4 : Print("Indicator shouldn't load its own data"); break;
        case -5 : Print("Load failed"); break;
        case 0 : Print("Loaded OK"); break;
        case 1 : Print("Loaded previously"); break;
        case 2 : Print("Loaded previously and built"); break;
        default : Print("Unknown result");
    }
//---
    datetime first_date;
    SeriesInfoInteger(InpLoadedSymbol,InpLoadedPeriod,SERIES_FIRSTDATE,first_date);
    int bars=Bars(InpLoadedSymbol,InpLoadedPeriod);
    Print("First date ",first_date," - ",bars," bars");
//---
}
//+-----+
//|                                     |
//+-----+

int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
{
    datetime first_date=0;
    datetime times[100];
//--- check symbol & period
    if(symbol==NULL || symbol=="") symbol=Symbol();
    if(period==PERIOD_CURRENT) period=Period();
//--- check if symbol is selected in the Market Watch
    if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
    {
        if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
        SymbolSelect(symbol,true);
    }
//--- check if data is present
    SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
    if(first_date>0 && first_date<=start_date) return(1);
//--- don't ask for load of its own data if it is an indicator
    if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Symbol()==symbol)
        return(-4);
//--- second attempt
    if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
    {
        //--- there is loaded data to build timeseries
    }
}

```



```

    if(first_date>0)
    {
        //--- force timeseries build
        CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
        //--- check date
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
    }
}

//--- max bars in chart from terminal options
int maxBars=TerminalInfoInteger(TERMINAL_MAXBARS);
//--- load symbol history info
datetime first_server_date=0;
while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date))
    Sleep(5);
//--- fix start date for loading
if(first_server_date>start_date) start_date=first_server_date;
if(first_date>0 && first_date<first_server_date)
    Print("Warning: first server date ",first_server_date," for ",symbol,
        " does not match to first series date ",first_date);
//--- load data step by step
int fail_cnt=0;
while(!IsStopped())
{
    //--- wait for timeseries build
    while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
        Sleep(5);
    //--- ask for built bars
    int bars=Bars(symbol,period);
    if(bars>0)
    {
        if(bars>=maxBars) return(-2);
        //--- ask for first date
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(0);
    }
    //--- copying of next part forces data loading
    int copied=CopyTime(symbol,period,bars,100,times);
    if(copied>0)
    {
        //--- check for data
        if(times[0]<=start_date) return(0);
        if(bars+copied>=maxBars) return(-2);
        fail_cnt=0;
    }
    else
    {
        //--- no more than 100 failed attempts
        fail_cnt++;
    }
}

```



```

        if(fail_cnt>=100) return(-5);
        Sleep(10);
    }
}

//--- stopped
return(-3);
}

//+-----+
//| Returns string value of the period |
//+-----+
string GetPeriodName(ENUM_TIMEFRAMES period)
{
    if(period==PERIOD_CURRENT) period=Period();
//---
    switch(period)
    {
        case PERIOD_M1: return("M1");
        case PERIOD_M2: return("M2");
        case PERIOD_M3: return("M3");
        case PERIOD_M4: return("M4");
        case PERIOD_M5: return("M5");
        case PERIOD_M6: return("M6");
        case PERIOD_M10: return("M10");
        case PERIOD_M12: return("M12");
        case PERIOD_M15: return("M15");
        case PERIOD_M20: return("M20");
        case PERIOD_M30: return("M30");
        case PERIOD_H1: return("H1");
        case PERIOD_H2: return("H2");
        case PERIOD_H3: return("H3");
        case PERIOD_H4: return("H4");
        case PERIOD_H6: return("H6");
        case PERIOD_H8: return("H8");
        case PERIOD_H12: return("H12");
        case PERIOD_D1: return("Daily");
        case PERIOD_W1: return("Weekly");
        case PERIOD_MN1: return("Monthly");
    }
//---
    return("unknown period");
}

```


SeriesInfoInteger

Returns information about the state of historical data. There are 2 variants of function calls.

Directly returns the property value.

```
long SeriesInfoInteger(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    ENUM_SERIES_INFO_INTEGER prop_id,  // property identifier
);
```

Returns true or false depending on the success of the function run.

```
bool SeriesInfoInteger(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    ENUM_SERIES_INFO_INTEGER prop_id,  // property ID
    long&           long_var          // variable for getting info
);
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

prop_id

[in] Identifier of the requested property, value of the [ENUM_SERIES_INFO_INTEGER](#) enumeration.

long_var

[out] Variable to which the value of the requested property is placed.

Return Value

In the first case, it returns value of the long type.

For the second case, it returns true, if the specified property is available and its value has been placed into long_var variable, otherwise it returns false. For more details about an [error](#), call [GetLastError\(\)](#).

Example:

```
void OnStart()
{
    //---
    Print("Total number of bars for the symbol-period at this moment = ",
        SeriesInfoInteger(Symbol(), 0, SERIES_BARS_COUNT));

    Print("The first date for the symbol-period at this moment = ",
        (datetime)SeriesInfoInteger(Symbol(), 0, SERIES_FIRSTDATE));
```



```
Print("The first date in the history for the symbol-period on the server = ",  
      (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));  
  
Print("Symbol data are synchronized = ",  
      (bool)SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED));  
}
```


Bars

Returns the number of bars count in the history for a specified symbol and period. There are 2 variants of functions calls.

Request all of the history bars

```
int Bars(  
    string          symbol_name,    // symbol name  
    ENUM_TIMEFRAMES timeframe      // period  
);
```

Request the history bars for the selected time interval

```
int Bars(  
    string          symbol_name,    // symbol name  
    ENUM_TIMEFRAMES timeframe,      // period  
    datetime        start_time,     // start date and time  
    datetime        stop_time      // end date and time  
);
```

Parameters

symbol_name
[in] Symbol name.

timeframe
[in] Period.

start_time
[in] Bar time corresponding to the first element.

stop_time
[in] Bar time corresponding to the last element.

Return Value

If the *start_time* and *stop_time* parameters are defined, the function returns the number of bars in the specified time interval, otherwise it returns the total number of bars.

Note

If data for the timeseries with specified parameters are not formed in the terminal by the time of the `Bars()` function call, or data of the timeseries are not [synchronized](#) with a trade server by the moment of the function call, the function returns a zero value.

When requesting the number of bars in a specified time interval, only bars with an open time falling within the interval are considered. For example, if the current day of the week is Saturday and the request is made for the number of W1 bars with *start_time*=last_tuesday and *stop_time*=last_friday, the function will return 0 since the open time of a W1 timeframe is always Sunday and not a single W1 bar falls within the specified interval.

Sample request for the number of all history bars:


```

int bars=Bars(_Symbol,_Period);
if(bars>0)
{
    Print("Number of bars in the terminal history for the symbol-period at the moment");
}
else //no available bars
{
    //--- data on the symbol might be not synchronized with data on the server
    bool synchronized=false;
    //--- loop counter
    int attempts=0;
    // make 5 attempts to wait for synchronization
    while(attempts<5)
    {
        if(SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED))
        {
            //--- synchronization done, exit
            synchronized=true;
            break;
        }
        //--- increase the counter
        attempts++;
        //--- wait 10 milliseconds till the next iteration
        Sleep(10);
    }
    //--- exit the loop after synchronization
    if(synchronized)
    {
        Print("Number of bars in the terminal history for the symbol-period at the moment");
        Print("The first date in the terminal history for the symbol-period at the moment");
        Print((datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));
        Print("The first date in the history for the symbol on the server = ",
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
    }
    //--- synchronization of data didn't happen
    else
    {
        Print("Failed to get number of bars for ",_Symbol);
    }
}

```

Sample request for the number of bars in the specified interval:

```

int n;
datetime date1 = D'2016.09.02 23:55'; // Friday
datetime date2 = D'2016.09.05 00:00'; // Monday
datetime date3 = D'2016.09.08 00:00'; // Thursday
//---
n=Bars(_Symbol,PERIOD_H1,D'2016.09.02 02:05',D'2016.09.02 10:55');

```



```
Print("Number of bars: ",n); // Output: "Number of bars: 8", H2 bar is considered :
n=Bars(_Symbol,PERIOD_D1,date1,date2);
Print("Number of bars: ",n); // Output: "Number of bars: 1", since an open time of
n=Bars(_Symbol,PERIOD_W1,date2,date3);
Print("Number of bars: ",n); // Output: "Number of bars: 0", since not a single W1
```

See also

[Event Handling Functions](#)

BarsCalculated

Returns the number of calculated data for the specified indicator.

```
int BarsCalculated(  
    int      indicator_handle,    // indicator handle  
);
```

Parameters

indicator_handle

[in] The indicator handle, returned by the corresponding indicator function.

Return Value

Returns the amount of calculated data in the indicator buffer or -1 in the case of error (data not calculated yet)

Note

The function is useful when it's necessary to get the indicator data immediately after its creation (indicator handle is available).

Example:

```
void OnStart()  
{  
    double Ups[];  
    //--- set timeseries ordering for the arrays  
    ArraySetAsSeries(Ups,true);  
    //--- create handle for the Fractal Indicator  
    int FractalsHandle=iFractals(NULL,0);  
    //--- reset the error code  
    ResetLastError();  
    //--- try to copy the indicator values  
    int i,copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);  
    if(copied<=0)  
    {  
        Sleep(50);  
        for(i=0;i<100;i++)  
        {  
            if(BarsCalculated(FractalsHandle)>0)  
                break;  
            Sleep(50);  
        }  
        copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);  
        if(copied<=0)  
        {  
            Print("Failed to copy upper fractals. Error = ",GetLastError(),  
                "i = ",i,"    copied = ",copied);  
            return;  
        }  
    }  
}
```



```
        else
            Print("Upper fractals copied",
                "i = ",i,"    copied = ",copied);
    }
    else Print("Upper fractals copied. ArraySize = ",ArraySize(Ups));
}
```


IndicatorCreate

The function returns the handle of a specified technical indicator created based on the array of parameters of [MqlParam](#) type.

```
int IndicatorCreate(
    string          symbol,           // symbol name
    ENUM_TIMEFRAMES period,          // timeframe
    ENUM_INDICATOR  indicator_type,  // indicator type from the enum
    int             parameters_cnt=0, // number of parameters
    const MqlParam& parameters_array[]=NULL, // array of parameters
);
```

Parameters

symbol

[in] Name of a symbol, on data of which the indicator is calculated. [NULL](#) means the current symbol.

period

[in] The value of the timeframe can be one of values of the [ENUM_TIMEFRAMES](#) enumeration, 0 means the current timeframe.

indicator_type

[in] Indicator type, can be one of values of the [ENUM_INDICATOR](#) enumeration.

parameters_cnt

[in] The number of parameters passed in the `parameters_array[]` array. The array elements have a special structure type [MqlParam](#). By default, zero - parameters are not passed. If you specify a non-zero number of parameters, the parameter `parameters_array` is obligatory. You can pass no more than 256 parameters.

parameters_array[]=NULL

[in] An array of [MqlParam](#) type, whose elements contain the type and value of each input parameter of a [technical indicator](#).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#).

Note

If the indicator handle of `IND_CUSTOM` type is created, the *type* field of the first element of the array of input parameters `parameters_array` must have the `TYPE_STRING` value of the [ENUM_DATATYPE](#) enumeration, and the *string_value* field of the first element must contain the name of the custom indicator. The custom indicator must be compiled (file with EX5 extension) and located in the directory MQL5/Indicators of the client terminal or in a subdirectory.

Indicators that require testing are defined automatically from the call of the `iCustom()` function, if the corresponding parameter is set through a [constant string](#). For all other cases (use of the [IndicatorCreate\(\)](#) function or use of a non-constant string in the parameter that sets the indicator name) the property [#property tester_indicator](#) is required:

```
#property tester_indicator "indicator_name.ex5"
```


If [the first form of the call](#) is used in a custom indicator, you can additionally indicate as the last parameter on what data it will be calculated when passing input parameters. If the "Apply to" parameter is not specified explicitly, the default calculation is based on the [PRICE_CLOSE](#) values.

Example:

```
void OnStart()
{
    MqlParam params[];
    int      h_MA, h_MACD;
    //--- create iMA("EURUSD", PERIOD_M15, 8, 0, MODE_EMA, PRICE_CLOSE);
    ArrayResize(params, 4);
    //--- set ma_period
    params[0].type      =TYPE_INT;
    params[0].integer_value=8;
    //--- set ma_shift
    params[1].type      =TYPE_INT;
    params[1].integer_value=0;
    //--- set ma_method
    params[2].type      =TYPE_INT;
    params[2].integer_value=MODE_EMA;
    //--- set applied_price
    params[3].type      =TYPE_INT;
    params[3].integer_value=PRICE_CLOSE;
    //--- create MA
    h_MA=IndicatorCreate("EURUSD", PERIOD_M15, IND_MA, 4, params);
    //--- create iMACD("EURUSD", PERIOD_M15, 12, 26, 9, h_MA);
    ArrayResize(params, 4);
    //--- set fast ma_period
    params[0].type      =TYPE_INT;
    params[0].integer_value=12;
    //--- set slow ma_period
    params[1].type      =TYPE_INT;
    params[1].integer_value=26;
    //--- set smooth period for difference
    params[2].type      =TYPE_INT;
    params[2].integer_value=9;
    //--- set indicator handle as applied_price
    params[3].type      =TYPE_INT;
    params[3].integer_value=h_MA;
    //--- create MACD based on moving average
    h_MACD=IndicatorCreate("EURUSD", PERIOD_M15, IND_MACD, 4, params);
    //--- use indicators
    //--- . . .
    //--- release indicators (first h_MACD)
    IndicatorRelease(h_MACD);
    IndicatorRelease(h_MA);
}
```


IndicatorParameters

Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters.

```
int IndicatorParameters(
    int          indicator_handle,    // indicator handle
    ENUM_INDICATOR& indicator_type,    // a variable for receiving the indicator type
    MqlParam&     parameters[]        // an array for receiving parameters
);
```

Parameters

indicator_handle

[in] The handle of the indicator, for which you need to know the number of parameters its is calculated on.

indicator_type

[out] A variable of the [ENUM_INDICATOR](#) type, into which the indicator type will be written.

parameters[]

[out] A dynamic array for receiving values of the [MqlParam](#) type, into which the list of indicator parameters will be written. The array size is returned by the `IndicatorParameters()` function.

Return Value

The number of input parameters of the indicator with the specified handle. In case of an error returns -1. For more details about the error call the [GetLastError\(\)](#) function.

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- The number of windows on the chart (at least one main window is always present)
    int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
    //--- Go through the chart windows
    for(int w=0;w<windows;w++)
    {
        //--- The number of indicators in this window/subwindow
        int total=ChartIndicatorsTotal(0,w);
        //--- Take all indicators in the window
        for(int i=0;i<total;i++)
        {
            //--- Get the short name of the indicator
            string name=ChartIndicatorName(0,w,i);
            //--- Get the indicator handle
            int handle=ChartIndicatorGet(0,w,name);
            //--- Add to log
        }
    }
}
```



```

PrintFormat("Window=%d,  indicator #%d,  handle=%d",w,i,handle);
//---
MqlParam parameters[];
ENUM_INDICATOR indicator_type;
int params=IndicatorParameters(handle,indicator_type,parameters);
//--- The header of the message
string par_info="Short name "+name+", type "
                +EnumToString(ENUM_INDICATOR(indicator_type))+"\r\n";

//---
for(int p=0;p<params;p++)
{
    par_info+=StringFormat("parameter %d: type=%s, long_value=%d, double_value=%d, string_value=%s",
                           p,
                           EnumToString((ENUM_DATATYPE)parameters[p].type),
                           parameters[p].integer_value,
                           parameters[p].double_value,
                           parameters[p].string_value
                           );

    }
    Print(par_info);
}
//--- Done for all indicators in the window
}
//---
}

```

See also

[ChartIndicatorGet\(\)](#)

IndicatorRelease

The function removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else.

```
bool IndicatorRelease(
    int      indicator_handle    // indicator handle
);
```

Return Value

Returns true in case of success, otherwise returns false.

Note

The function allows removing an indicator handle, if it's no longer needed, thus saving memory. The handle is removed immediately, the calculation block is deleted in some time (if it's not called anymore).

When working in the [strategy tester](#), the IndicatorRelease() function is not executed.

Example:

```
//+-----+
//|                                     Test_IndicatorRelease.mq5 |
//|                                     Copyright 2010, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input int          MA_Period=15;
input int          MA_shift=0;
input ENUM_MA_METHOD MA_smooth=MODE_SMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
//--- will store indicator handle
int MA_handle=INVALID_HANDLE;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create indicator handle
    MA_handle=iMA(Symbol(),0,MA_Period,MA_shift,MA_smooth,PRICE_CLOSE);
    //--- delete global variable
    if(GlobalVariableCheck("MA_value"))
        GlobalVariableDel("MA_value");
    //---
    return(INIT_SUCCEEDED);
}
```



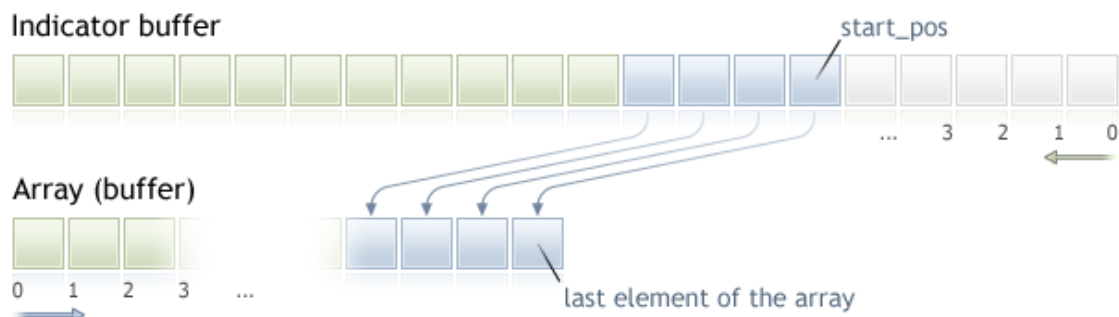
```

///| Expert tick function |
//+-----+
void OnTick()
{
//--- if the global variable value does not exist
if(!GlobalVariableCheck("MA_value"))
{
//--- obtain the indicator value in the last two bars
if(MA_handle!=INVALID_HANDLE)
{
//--- dynamic array for the indicator values
double values[];
if(CopyBuffer(MA_handle,0,0,2,values)==2 && values[0]!=EMPTY_VALUE)
{
//--- remember in the global variable value on the last but one bar
if(GlobalVariableSet("MA_value",values[0]))
{
//--- free the handle of the indicator
if(!IndicatorRelease(MA_handle))
Print("IndicatorRelease() failed. Error ",GetLastError());
else MA_handle=INVALID_HANDLE;
}
else
Print("GlobalVariableSet failed. Error ",GetLastError());
}
}
}
//---
}

```


CopyBuffer

Gets data of a specified buffer of a certain indicator in the necessary quantity.



Counting of elements of copied data (indicator buffer with the index buffer_num) from the starting position is performed from the present to the past, i.e., starting position of 0 means the current bar (indicator value for the current bar).

When copying the yet unknown amount of data, it is recommended to use a [dynamic array](#) as a buffer[] recipient buffer, because the CopyBuffer() function tries to allocate the size of the receiving array to the size of the copied data. If an indicator buffer (array that is pre-allocated for storing indicator values by the [SetIndexBufer\(\)](#) function) is used as the buffer[] recipient array, partial copying is allowed. An example can be found in the Awesome_Oscillator.mql5 custom indicator in the standard terminal package.

If you need to make a partial copy of the indicator values into another array (non-indicator buffer), you should use an intermediate array, to which the desired number is copied. After that conduct the element-wise copying of the required number of values into the required places of a receiving array from this intermediate one.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyBuffer(
    int      indicator_handle,    // indicator handle
    int      buffer_num,         // indicator buffer number
    int      start_pos,          // start position
    int      count,              // amount to copy
    double   buffer[])           // target array to copy
);
```

Call by the start date and the number of required elements

```
int CopyBuffer(
    int      indicator_handle,    // indicator handle
```



```

int      buffer_num,           // indicator buffer number
datetime start_time,          // start date and time
int      count,               // amount to copy
double   buffer[]             // target array to copy
);

```

Call by the start and end dates of a required time interval

```

int CopyBuffer(
    int      indicator_handle,   // indicator handle
    int      buffer_num,        // indicator buffer number
    datetime start_time,        // start date and time
    datetime stop_time,         // end date and time
    double   buffer[]           // target array to copy
);

```

Parameters

indicator_handle

[in] The indicator handle, returned by the corresponding indicator function.

buffer_num

[in] The indicator buffer number.

start_pos

[in] The position of the first element to copy.

count

[in] Data count to copy.

start_time

[in] Bar time, corresponding to the first element.

stop_time

[in] Bar time, corresponding to the last element.

buffer[]

[out] Array of [double](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration.

Example:

```
//+-----+
//|                                     TestCopyBuffer3.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot MA
#property indicator_label1 "MA"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool      AsSeries=true;
input int       period=15;
input ENUM_MA_METHOD smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int       shift=0;
//--- indicator buffers
double          MABuffer[];
int             ma_handle;
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
Print("Parameter AsSeries = ",AsSeries);
Print("Indicator buffer after SetIndexBuffer() is a timeseries = ",
      ArrayGetAsSeries(MABuffer));
//--- set short indicator name
IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//--- set AsSeries (depends on input parameter)
ArraySetAsSeries(MABuffer,AsSeries);
Print("Indicator buffer after ArraySetAsSeries(MABuffer,true); is a timeseries = ",
      ArrayGetAsSeries(MABuffer));
//---
ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
return(INIT_SUCCEEDED);
}
```



```
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- check if all data calculated
    if(BarsCalculated(ma_handle)<rates_total) return(0);
    //--- we can copy not all data
    int to_copy;
    if(prev_calculated>rates_total || prev_calculated<=0) to_copy=rates_total;
    else
    {
        to_copy=rates_total-prev_calculated;
        //--- last value is always copied
        to_copy++;
    }
    //--- try to copy
    if(CopyBuffer(ma_handle,0,0,to_copy,MABuffer)<=0) return(0);
    //--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
```

The above example illustrates how an indicator buffer is filled out with the values of another indicator buffer from the indicator on the same symbol/period.

See a detailed example of history requesting data in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

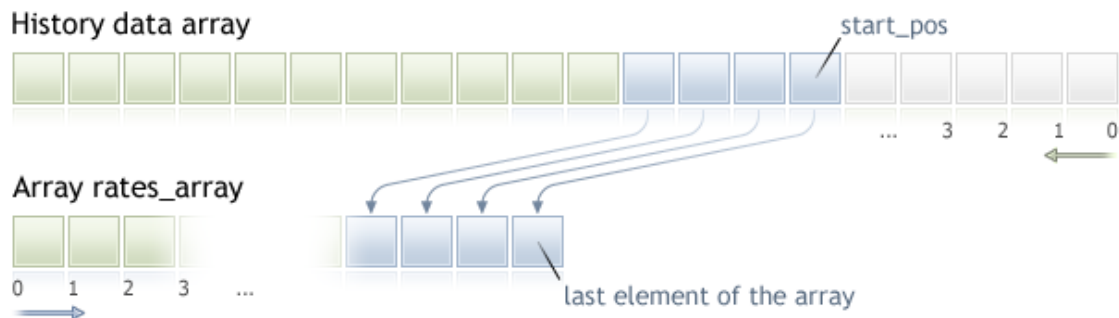
- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

See also

[Properties of Custom Indicators](#), [SetIndexBuffer](#)

CopyRates

Gets history data of [MqlRates](#) structure of a specified symbol-period in specified quantity into the `rates_array` array. The elements ordering of the copied data is from present to the past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyRates(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,         // start position
    int             count,             // data count to copy
    MqlRates        rates_array[]     // target array to copy
);
```

Call by the start date and the number of required elements

```
int CopyRates(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,             // data count to copy
    MqlRates        rates_array[]     // target array to copy
);
```

Call by the start and end dates of a required time interval

```
int CopyRates(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,       // start date and time
datetime    stop_time,        // end date and time
MqlRates    rates_array[]     // target array to copy
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_time

[in] Bar time for the first element to copy.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

rates_array[]

[out] Array of [MqlRates](#) type.

Return Value

Returns the number of copied elements or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

Example:

```
void OnStart()
{
//---
    MqlRates rates[];
    ArraySetAsSeries(rates,true);
    int copied=CopyRates(Symbol(),0,0,100,rates);
    if(copied>0)
    {
        Print("Bars copied: "+copied);
        string format="open = %G, high = %G, low = %G, close = %G, volume = %d";
        string out;
        int size=fmin(copied,10);
        for(int i=0;i<size;i++)
        {
            out=i+": "+TimeToString(rates[i].time);
            out=out+" "+StringFormat(format,
                                     rates[i].open,
                                     rates[i].high,
                                     rates[i].low,
                                     rates[i].close,
                                     rates[i].tick_volume);

            Print(out);
        }
    }
    else Print("Failed to get history data for the symbol ",Symbol());
}
```

See a detailed example of requesting history data in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),

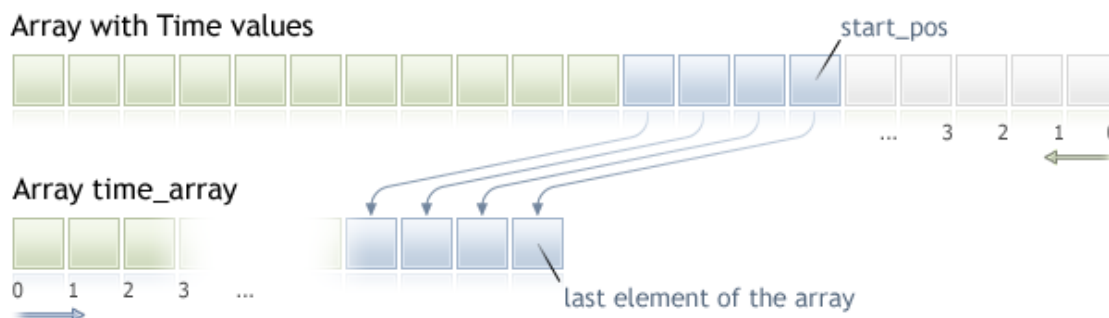
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

See also

[Structures and Classes](#), [TimeToString](#), [StringFormat](#)

CopyTime

The function gets to time_array history data of bar opening time for the specified symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyTime(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,       // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    datetime        time_array[]      // target array to copy open times
);
```

Call by the start date and the number of required elements

```
int CopyTime(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,       // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    datetime        time_array[]      // target array to copy open times
);
```

Call by the start and end dates of a required time interval

```
int CopyTime(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
datetime    time_array[]     // target array to copy open times
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time corresponding to the last element to copy.

time_array[]

[out] Array of [datetime](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

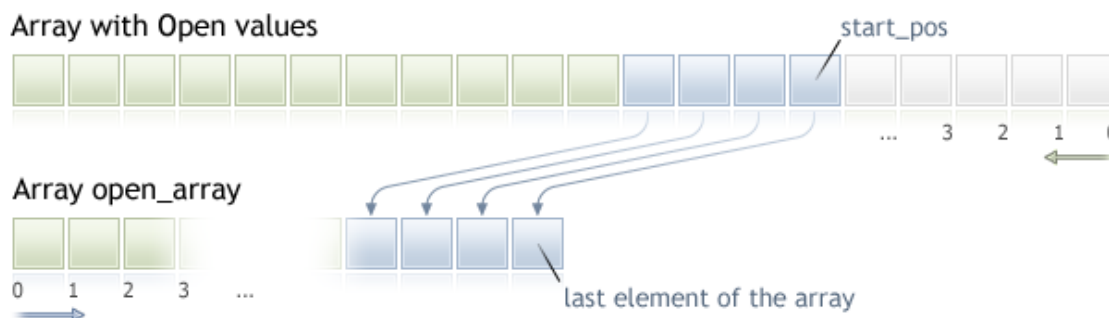
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

See a detailed example of requesting history data in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyOpen

The function gets into `open_array` the history data of bar open prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyOpen(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,         // start position
    int             count,             // data count to copy
    double          open_array[]      // target array to copy open prices
);
```

Call by the start date and the number of required elements

```
int CopyOpen(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,             // data count to copy
    double          open_array[]      // target array for bar open prices
);
```

Call by the start and end dates of a required time interval

```
int CopyOpen(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
double      open_array[]     // target array for bar open values
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] The start time for the last element to copy.

open_array[]

[out] Array of [double](#) type.

Return Value

Returns the number of element in the array or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

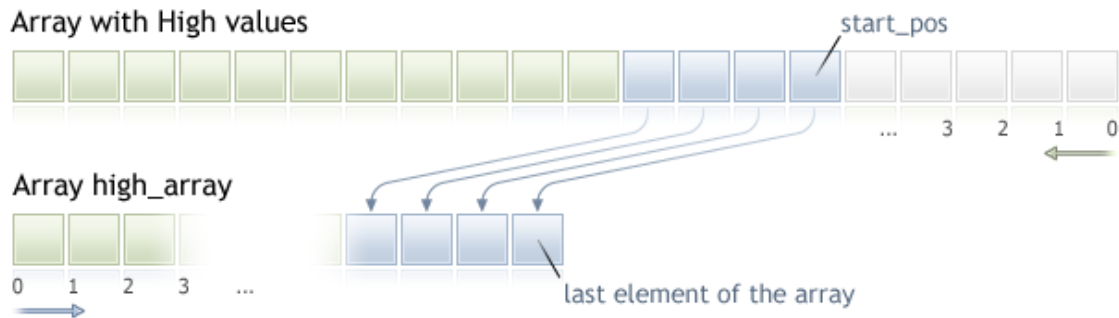
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

See a detailed example of requesting history data in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyHigh

The function gets into `high_array` the history data of highest bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyHigh(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    double          high_array[]      // target array to copy
);
```

Call by the start date and the number of required elements

```
int CopyHigh(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    double          high_array[]      // target array to copy
);
```

Call by the start and end dates of a required time interval

```
int CopyHigh(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
double      high_array[]      // target array to copy
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

high_array[]

[out] Array of [double](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

Example:

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An example for output of the High[i] and Low[i]"
#property description "for a random chosen bars"

double High[],Low[];
//+-----+
//| Get Low for specified bar index |
//+-----+
double iLow(string symbol,ENUM_TIMEFRAMES timeframe,int index)
{
    double low=0;
    ArraySetAsSeries(Low,true);
    int copied=CopyLow(symbol,timeframe,0,Bars(symbol,timeframe),Low);
    if(copied>0 && index<copied) low=Low[index];
    return(low);
}
//+-----+
//| Get the High for specified bar index |
//+-----+
double iHigh(string symbol,ENUM_TIMEFRAMES timeframe,int index)
{
    double high=0;
    ArraySetAsSeries(High,true);
    int copied=CopyHigh(symbol,timeframe,0,Bars(symbol,timeframe),High);
    if(copied>0 && index<copied) high=High[index];
    return(high);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{

```



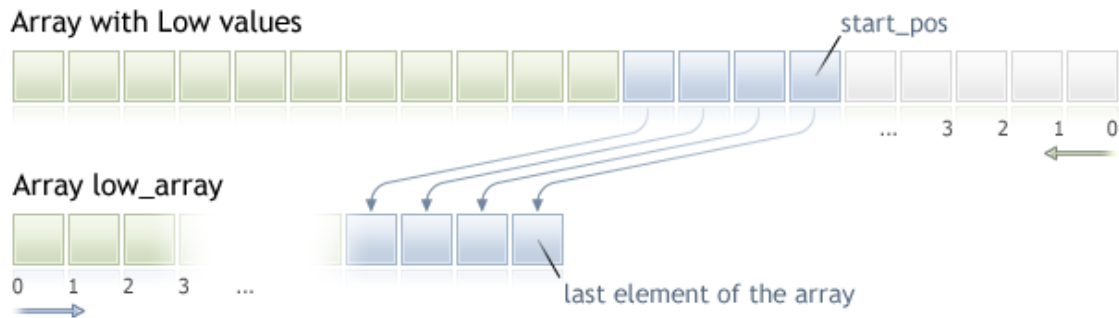
```
//--- on every tick we output the High and Low values for the bar with index,  
//--- that is equal to the second, on which tick arrived  
datetime t=TimeCurrent();  
int sec=t%60;  
printf("High[%d] = %G Low[%d] = %G",  
        sec,iHigh(Symbol(),0,sec),  
        sec,iLow(Symbol(),0,sec));  
}
```

See a detailed example of requesting history data in the [Methods of Object Binding](#) section. The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyLow

The function gets into `low_array` the history data of minimal bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyLow(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    double          low_array[]       // target array to copy
);
```

Call by the start date and the number of required elements

```
int CopyLow(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    double          low_array[]       // target array to copy
);
```

Call by the start and end dates of a required time interval

```
int CopyLow(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
double      low_array[]      // target array to copy
);

```

Parameters

symbol_name

[in] Symbol.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] Bar time, corresponding to the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

low_array[]

[out] Array of [double](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

See a detailed example of requesting history data in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

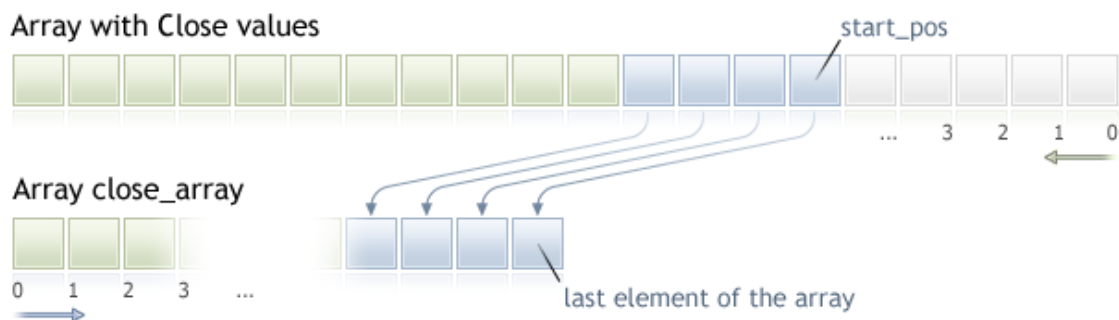
- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

See also

[CopyHigh](#)

CopyClose

The function gets into `close_array` the history data of bar close prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyClose(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    double          close_array[]     // target array to copy
);
```

Call by the start date and the number of required elements

```
int CopyClose(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    double          close_array[]     // target array to copy
);
```

Call by the start and end dates of a required time interval

```
int CopyClose(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
double      close_array[]     // target array to copy
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

close_array[]

[out] Array of [double](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

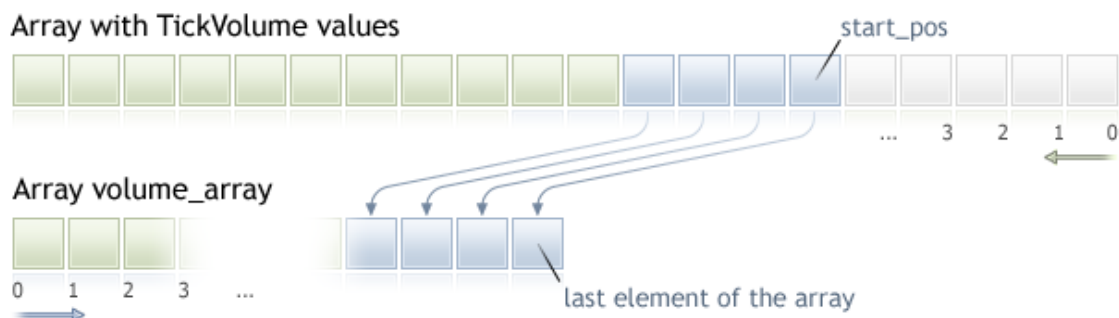
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

See a detailed example of history data requesting in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyTickVolume

The function gets into `volume_array` the history data of tick volumes for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyTickVolume(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    long            volume_array[]    // target array for tick volumes
);
```

Call by the start date and the number of required elements

```
int CopyTickVolume(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    long            volume_array[]    // target array for tick volumes
);
```

Call by the start and end dates of a required time interval

```
int CopyTickVolume(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
long        volume_array[]    // target array for tick volumes
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

volume_array[]

[out] Array of [long](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

Example:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot TickVolume
#property indicator_label1 "TickVolume"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 C'143,188,139'
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int bars=3000;
//--- indicator buffers
double TickVolumeBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,TickVolumeBuffer,INDICATOR_DATA);
IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
```



```

        const int &spread[]
    {
        //---
        if (prev_calculated==0)
        {
            long timeseries[];
            ArraySetAsSeries(timeseries,true);
            int prices=CopyTickVolume(Symbol(),0,0,bars,timeseries);
            for(int i=0;i<rates_total-prices;i++) TickVolumeBuffer[i]=0.0;
            for(int i=0;i<prices;i++) TickVolumeBuffer[rates_total-1-i]=timeseries[prices-1-i];
            Print("We have received the following number of TickVolume values: "+prices);
        }
        else
        {
            long timeseries[];
            int prices=CopyTickVolume(Symbol(),0,0,1,timeseries);
            TickVolumeBuffer[rates_total-1]=timeseries[0];
        }
        //--- return value of prev_calculated for next call
        return(rates_total);
    }

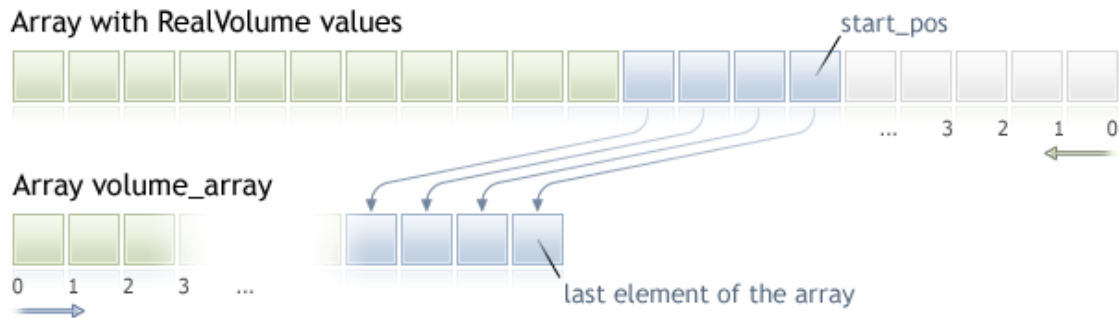
```

See a detailed example of history data requesting in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyRealVolume

The function gets into `volume_array` the history data of trade volumes for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopyRealVolume(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    long            volume_array[]    // target array for volumes values
);
```

Call by the start date and the number of required elements

```
int CopyRealVolume(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    long            volume_array[]    // target array for volumes values
);
```

Call by the start and end dates of a required time interval

```
int CopyRealVolume(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
long        volume_array[]    // target array for volumes values
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

volume_array[]

[out] Array of [long](#) type.

Return Value

Returns the copied data count or -1 in the case of [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

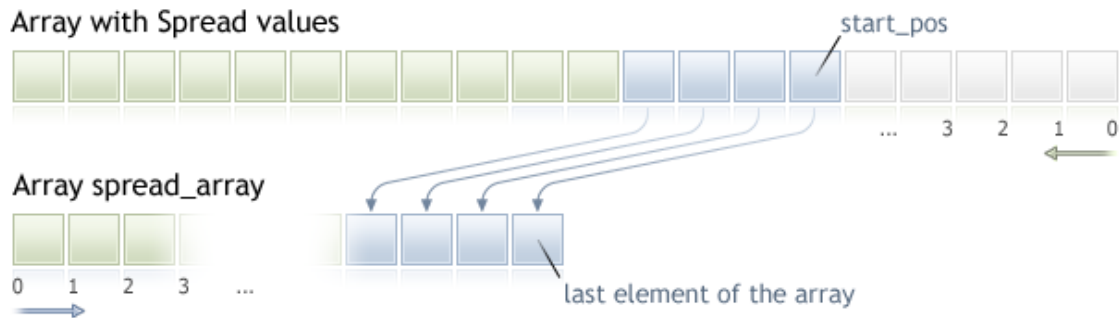
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

See an example of history data requesting in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopySpread

The function gets into `spread_array` the history data of spread values for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use [dynamic array](#) as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a [statically allocated buffer](#), in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - `as_series=true` or `as_series=false`. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int CopySpread(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    int             start_pos,        // start position
    int             count,            // data count to copy
    int             spread_array[]    // target array for spread values
);
```

Call by the start date and the number of required elements

```
int CopySpread(
    string          symbol_name,      // symbol name
    ENUM_TIMEFRAMES timeframe,        // period
    datetime        start_time,       // start date and time
    int             count,            // data count to copy
    int             spread_array[]    // target array for spread values
);
```

Call by the start and end dates of a required time interval

```
int CopySpread(
```



```

string      symbol_name,      // symbol name
ENUM_TIMEFRAMES timeframe,    // period
datetime    start_time,      // start date and time
datetime    stop_time,       // stop date and time
int         spread_array[]    // target array for spread values
);

```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

spread_array[]

[out] Array of [int](#) type.

Return Value

Returns the copied data count or -1 in case of an [error](#).

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside [TERMINAL_MAXBARS](#) (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, [downloading from the server](#) will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos=0* and *count=1*.

Example:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Spread
#property indicator_label1 "Spread"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int bars=3000;
//--- indicator buffers
double SpreadBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0, SpreadBuffer, INDICATOR_DATA);
IndicatorSetInteger(INDICATOR_DIGITS, 0);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
```



```

        const int &spread[])
    {
//---
    if (prev_calculated==0)
    {
        int spread_int[];
        ArraySetAsSeries (spread_int,true);
        int spreads=CopySpread(Symbol(),0,0,bars,spread_int);
        Print("We have received the following number of Spread values: ",spreads);
        for (int i=0;i<spreads;i++)
        {
            SpreadBuffer[rates_total-1-i]=spread_int[i];
            if (i<=30) Print("spread["+i+"] = ",spread_int[i]);
        }
    }
    else
    {
        double Ask,Bid;
        Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
        Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
        Comment("Ask = ",Ask," Bid = ",Bid);
        SpreadBuffer[rates_total-1]=(Ask-Bid)/Point();
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

See an example of history data requesting in section [Methods of Object Binding](#). The script available in that section shows how to get the values of indicator [iFractals](#) on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following [styles](#):

- [DRAW_SECTION](#),
- [DRAW_ARROW](#),
- [DRAW_ZIGZAG](#),
- [DRAW_COLOR_SECTION](#),
- [DRAW_COLOR_ARROW](#),
- [DRAW_COLOR_ZIGZAG](#).

CopyTicks

The function receives ticks in the [MqlTick](#) format into `ticks_array`. In this case, ticks are indexed from the past to the present, i.e. the 0 indexed tick is the oldest one in the array. For tick analysis, check the *flags* field, which shows what exactly has changed in the tick.

```
int CopyTicks(
    string          symbol_name,           // Symbol name
    MqlTick&        ticks_array[],        // Tick receiving array
    uint            flags=COPY_TICKS_ALL,  // The flag that determines the type of request
    ulong           from=0,               // The date from which you want to request ticks
    uint            count=0               // The number of ticks that you want to receive
);
```

Parameters

symbol_name

[in] Symbol.

ticks_array

[out] An array of the [MqlTick](#) type for receiving ticks.

flags

[in] A flag to define the type of the requested ticks. [COPY_TICKS_INFO](#) - ticks with Bid and/or Ask changes, [COPY_TICKS_TRADE](#) - ticks with changes in Last and Volume, [COPY_TICKS_ALL](#) - all ticks. For any type of request, the values of the previous tick are added to the remaining fields of the `MqlTick` structure.

from

[in] The date from which you want to request ticks. In milliseconds since 1970.01.01. If *from*=0, the last *count* ticks will be returned.

count

[in] The number of requested ticks. If the 'from' and 'count' parameters are not specified, all available recent ticks (but not more than 2000) will be written to `ticks_array[]`.

Returned value

The number of copied tick or -1 in case of an [error](#).

Note

The `CopyTicks()` function allows requesting and analyzing all received ticks. The first call of `CopyTicks()` initiates synchronization of the symbol's tick database stored on the hard disk. If the local database does not provide all the requested ticks, then missing ticks will be automatically downloaded from the trade server. Ticks beginning with the *from* date specified in `CopyTicks()` till the current moment will be synchronized. After that, all ticks arriving for this symbol will be added to the tick database thus keeping it in the synchronized state.

If the *from* and *count* parameters are not specified, all available recent ticks (but not more than 2000) will be written to `ticks_array[]`. The *flags* parameter allows specifying the type of required ticks.

COPY_TICKS_INFO - ticks with Bid and/or Ask price changes are returned. Data of other fields will also be added. For example, if only the Bid has changed, the *ask* and *volume* fields will be filled with last known values. To find out exactly what has changed, analyze the *flags* field, which will have the value of TICK_FLAG_BID and/or TICK_FLAG_ASK. If a tick has zero values of the Bid and Ask prices, and the flags show that these data have changed (*flags*=TICK_FLAG_BID|TICK_FLAG_ASK), this means that the order book (Market Depth) is empty. In other words, there are no buy and sell orders.

COPY_TICKS_TRADE - ticks with the Last price and volume changes are returned. Data of other fields will also be added, i.e. last known values of Bid and Ask will be specified in the appropriate fields. To find out exactly what has changed, analyze the *flags* field, which will have the TICK_FLAG_LAST and TICK_FLAG_VOLUME value.

COPY_TICKS_ALL - all ticks with any change are returned. Unchanged fields will be filled with last known values.

Call of CopyTicks() with the COPY_TICKS_ALL flag immediately returns all ticks from the request interval, while calls in other modes require some time to process and select ticks, therefore they do not provide significant speed advantage.

When requesting ticks (either **COPY_TICKS_INFO** or **COPY_TICKS_TRADE**), every tick contains full price information as of the time of the tick (*bid*, *ask*, *last* and *volume*). This feature is provided for an easier analysis of the trade state at the time of each tick, so there is no need to request a deep tick history and search for the values of other fields.

In indicators, the CopyTicks() function returns the result: when called from an indicator, CopyTick() immediately returns all available ticks of a symbol, and will launch synchronization of the tick database, if available data is not enough. All indicators in one symbol operate in one common thread, so the indicator cannot wait for the completion of synchronization. After synchronization, CopyTicks() will return all requested ticks during the next call. In indicators, the [OnCalculate\(\)](#) function is called after the arrival of each tick.

CopyTicks() can wait for the result for 45 seconds in Expert Advisors and scripts: as distinct from indicators, every Expert Advisor and script operate in a separate thread, and therefore can wait 45 seconds till the completion of synchronization. If the required amount of ticks fails to be synchronized during this time, CopyTicks() will return available ticks by timeout and will continue synchronization. [OnTick\(\)](#) in Expert Advisor is not a handler of every tick, it only notifies an Expert Advisor about changes in the market. It can be a batch of changes: the terminal can simultaneously make a few ticks, but OnTick() will be called only once to notify the EA of the latest market state.

The rate of data return: the terminal stores in the fast access cache 4,096 last ticks for each instrument (65,536 ticks for symbols with a running Market Depth). If requested ticks for the current trading session are beyond the cache, CopyTicks() calls the ticks stored in the terminal memory. These requests require more time for execution. The slowest requests are those requesting ticks for other days, since the data is read from the disk in this case.

Example:

```
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs

//--- Requesting 100 million ticks to be sure we receive the entire tick history
input int          getticks=100000000; // The number of required ticks
```



```

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
    int      attempts=0;      // Count of attempts
    bool      success=false;  // The flag of a successful copying of ticks
    MqlTick tick_array[];     // Tick receiving array
    MqlTick lasttick;         // To receive last tick data
    SymbolInfoTick(_Symbol,lasttick);
//--- Make 3 attempts to receive ticks
    while(attempts<3)
    {
        //--- Measuring start time before receiving the ticks
        uint start=GetTickCount();
//--- Requesting the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
        int received=CopyTicks(_Symbol,tick_array,COPY_TICKS_ALL,1,getticks);
        if(received!=-1)
        {
            //--- Showing information about the number of ticks and spent time
            PrintFormat("%s: received %d ticks in %d ms",_Symbol,received,GetTickCount()-start);
            //--- If the tick history is synchronized, the error code is equal to zero
            if(GetLastError()==0)
            {
                success=true;
                break;
            }
            else
                PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms",_Symbol,received,GetTickCount()-start,_LastError);
        }
        //--- Counting attempts
        attempts++;
        //--- A one-second pause to wait for the end of synchronization of the tick data
        Sleep(1000);
    }
//--- Receiving the requested ticks from the beginning of the tick history failed in three attempts
    if(!success)
    {
        PrintFormat("Error! Failed to receive %d ticks of %s in three attempts",getticks,_Symbol);
        return;
    }
    int ticks=ArraySize(tick_array);
//--- Showing the time of the first tick in the array
    datetime firstticktime=tick_array[ticks-1].time;
    PrintFormat("Last tick time = %s.%03I64u",
        TimeToString(firstticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_array[ticks-1].time_msec);
//--- Show the time of the last tick in the array

```



```

datetime lastticktime=tick_array[0].time;
PrintFormat("First tick time = %s.%03I64u",
            TimeToString(lastticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_ar

//--- Getting the beginning and the end of the day from the tick
MqlDateTime today;
TimeToStruct(lasttick.time,today);
today.hour=0;
today.min=0;
today.sec=0;
datetime startday=StructToTime(today);
datetime endday=startday+24*60*60;
//--- Showing the first 100 ticks of the last day
int counter=0;
for(int i=0;i<ticks;i++)
{
    datetime time=tick_array[i].time;
    if((time>=startday) && (time<endday) && counter<100)
    {
        counter++;
        PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
    }
}
//--- Showing the first 100 deals of the last day
counter=0;
for(int i=0;i<ticks;i++)
{
    datetime time=tick_array[i].time;
    if((time>=startday) && (time<endday) && counter<100)
    {
        if(((tick_array[i].flags&TICK_FLAG_BUY)==TICK_FLAG_BUY) || ((tick_array[i].f
        {
            counter++;
            PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
        }
    }
}

}

//+-----+
//| Returns the string description of a tick |
//+-----+
string GetTickDescription(MqlTick &tick)
{
    string desc=StringFormat("%s.%03d ",
                             TimeToString(tick.time),tick.time_msc%1000);
//--- Checking flags
bool buy_tick=((tick.flags&TICK_FLAG_BUY)==TICK_FLAG_BUY);
bool sell_tick=((tick.flags&TICK_FLAG_SELL)==TICK_FLAG_SELL);
bool ask_tick=((tick.flags&TICK_FLAG_ASK)==TICK_FLAG_ASK);

```



```

bool bid_tick=((tick.flags&TICK_FLAG_BID)==TICK_FLAG_BID);
bool last_tick=((tick.flags&TICK_FLAG_LAST)==TICK_FLAG_LAST);
bool volume_tick=((tick.flags&TICK_FLAG_VOLUME)==TICK_FLAG_VOLUME);
//--- Checking trading flags in a tick first
if(buy_tick || sell_tick)
{
    //--- Forming an output for the trading tick
    desc=desc+(buy_tick?StringFormat("Buy Tick: Last=%G Volume=%d ",tick.last,tick.v
    desc=desc+(sell_tick?StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last,tick
    desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask): "");
    desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask): "");
    desc=desc+"(Trade tick)";
}
else
{
    //--- Form a different output for an info tick
    desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask): "");
    desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask): "");
    desc=desc+(last_tick?StringFormat("Last=%G ",tick.last): "");
    desc=desc+(volume_tick?StringFormat("Volume=%d ",tick.volume): "");
    desc=desc+"(Info tick)";
}
//--- Returning tick description
return desc;
}
//+-----+
/* Example of the output
Si-12.16: received 11048387 ticks in 4937 ms
Last tick time = 2016.09.26 18:32:59.775
First tick time = 2015.06.18 09:45:01.000
1. 2016.09.26 09:45.249 Ask=65370 Bid=65370 (Info tick)
2. 2016.09.26 09:47.420 Ask=65370 Bid=65370 (Info tick)
3. 2016.09.26 09:50.893 Ask=65370 Bid=65370 (Info tick)
4. 2016.09.26 09:51.827 Ask=65370 Bid=65370 (Info tick)
5. 2016.09.26 09:53.810 Ask=65370 Bid=65370 (Info tick)
6. 2016.09.26 09:54.491 Ask=65370 Bid=65370 (Info tick)
7. 2016.09.26 09:55.913 Ask=65370 Bid=65370 (Info tick)
8. 2016.09.26 09:59.350 Ask=65370 Bid=65370 (Info tick)
9. 2016.09.26 09:59.678 Bid=65370 (Info tick)
10. 2016.09.26 10:00.000 Sell Tick: Last=65367 Volume=3 (Trade tick)
11. 2016.09.26 10:00.000 Sell Tick: Last=65335 Volume=45 (Trade tick)
12. 2016.09.26 10:00.000 Sell Tick: Last=65334 Volume=95 (Trade tick)
13. 2016.09.26 10:00.191 Sell Tick: Last=65319 Volume=1 (Trade tick)
14. 2016.09.26 10:00.191 Sell Tick: Last=65317 Volume=1 (Trade tick)
15. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=1 (Trade tick)
16. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=10 (Trade tick)
17. 2016.09.26 10:00.191 Sell Tick: Last=65315 Volume=5 (Trade tick)
18. 2016.09.26 10:00.191 Sell Tick: Last=65313 Volume=3 (Trade tick)
19. 2016.09.26 10:00.191 Sell Tick: Last=65307 Volume=25 (Trade tick)

```



```
20. 2016.09.26 10:00.191 Sell Tick: Last=65304 Volume=1 (Trade tick)
21. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=1 (Trade tick)
22. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=10 (Trade tick)
23. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=5 (Trade tick)
24. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=1 (Trade tick)
25. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=6 (Trade tick)
26. 2016.09.26 10:00.191 Sell Tick: Last=65299 Volume=1 (Trade tick)
27. 2016.09.26 10:00.191 Bid=65370 (Info tick)
28. 2016.09.26 10:00.232 Ask=65297 (Info tick)
29. 2016.09.26 10:00.276 Sell Tick: Last=65291 Volume=31 (Trade tick)
30. 2016.09.26 10:00.276 Sell Tick: Last=65290 Volume=1 (Trade tick)
*/
```

See also

[SymbolInfoTick](#), [Structure for Current Prices](#), [OnTick\(\)](#)

CopyTicksRange

The function receives ticks in the [MqlTick](#) format within the specified date range to ticks_array. Indexing goes from the past to the present meaning that a tick with the index 0 is the oldest one in the array. For tick analysis, check the *flags* field, which shows what exactly has changed.

```
int CopyTicksRange(
    const string      symbol_name,           // symbol name
    MqlTick&          ticks_array[],         // tick receiving array
    uint              flags=COPY_TICKS_ALL,  // flag that defines the type of the ticks
    ulong             from_msc=0,           // date, starting from which ticks are req
    ulong             to_msc=0              // date, up to which ticks are requested
);
```

Parameters

symbol_name

[in] Symbol.

ticks_array

[out] [MqlTick](#) static or dynamic array for receiving ticks. If the static array cannot hold all the ticks from the requested time interval, the maximum possible amount of ticks is received. In this case, the function generates the error [ERR_HISTORY_SMALL_BUFFER](#) (4407) .

flags

[in] A flag to define the type of the requested ticks. [COPY_TICKS_INFO](#) - ticks with Bid and/or Ask changes, [COPY_TICKS_TRADE](#) - ticks with changes in Last and Volume, [COPY_TICKS_ALL](#) - all ticks. For any type of request, the values of the previous tick are added to the remaining fields of the MqlTick structure.

from_msc

[in] The date, from which you want to request ticks. In milliseconds since 1970.01.01. If the *from_msc* parameter is not specified, ticks from the beginning of the history are sent. Ticks with the time >= from_msc are sent.

to_msc

[in] The date, up to which you want to request ticks. In milliseconds since 01.01.1970. Ticks with the time <= to_msc are sent. If the *to_msc* parameter is not specified, all ticks up to the end of the history are sent.

Return Value

The number of copied tick or -1 in case of an error. [GetLastError\(\)](#) is able to return the following errors:

- [ERR_HISTORY_TIMEOUT](#) - ticks synchronization waiting time is up, the function has sent all it had.
- [ERR_HISTORY_SMALL_BUFFER](#) - static buffer is too small. Only the amount the array can store has been sent.
- [ERR_NOT_ENOUGH_MEMORY](#) - insufficient memory for receiving a history from the specified range to the dynamic tick array. Failed to allocate enough memory for the tick array.

Note

The `CopyTicksRange()` function is used for requesting ticks strictly from a specified range, for example, from a certain day in history. At the same time, `CopyTicks()` allows specifying only a start date, for example - receive all ticks from the beginning of the month till the current moment.

See also

[SymbolInfoTick](#), [Structure for Current Prices](#), [OnTick](#), [CopyTicks](#)

Chart Operations

These are functions for working with charts. All chart operations are allowed in Expert Advisors and scripts only.

The functions defining the chart properties are actually used for sending change commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. The changes are implemented to the chart when handling the queue of the chart events.

Thus, do not expect an immediate visual update of the chart after calling these functions. Generally, the chart is updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use [ChartRedraw\(\)](#) function to forcefully update the chart.

Function	Action
ChartApplyTemplate	Applies a specific template from a specified file to the chart
ChartSaveTemplate	Saves current chart settings in a template with a specified name
ChartWindowFind	Returns the number of a subwindow where an indicator is drawn
ChartTimePriceToXY	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates
ChartXYToTimePrice	Converts the X and Y coordinates on a chart to the time and price values
ChartOpen	Opens a new chart with the specified symbol and period
ChartClose	Closes the specified chart
ChartFirst	Returns the ID of the first chart of the client terminal
ChartNext	Returns the chart ID of the chart next to the specified one
ChartSymbol	Returns the symbol name of the specified chart
ChartPeriod	Returns the period value of the specified chart
ChartRedraw	Calls a forced redrawing of a specified chart
ChartSetDouble	Sets the double value for a corresponding property of the specified chart
ChartSetInteger	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart

<u>ChartSetString</u>	Sets the string value for a corresponding property of the specified chart
<u>ChartGetDouble</u>	Returns the double value property of the specified chart
<u>ChartGetInteger</u>	Returns the integer value property of the specified chart
<u>ChartGetString</u>	Returns the string value property of the specified chart
<u>ChartNavigate</u>	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart
<u>ChartID</u>	Returns the ID of the current chart
<u>ChartIndicatorAdd</u>	Adds an indicator with the specified handle into a specified chart window
<u>ChartIndicatorDelete</u>	Removes an indicator with a specified name from the specified chart window
<u>ChartIndicatorGet</u>	Returns the handle of the indicator with the specified short name in the specified chart window
<u>ChartIndicatorName</u>	Returns the short name of the indicator by the number in the indicators list on the specified chart window
<u>ChartIndicatorsTotal</u>	Returns the number of all indicators applied to the specified chart window.
<u>ChartWindowOnDropped</u>	Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to
<u>ChartPriceOnDropped</u>	Returns the price coordinate of the chart point the Expert Advisor or script has been dropped to
<u>ChartTimeOnDropped</u>	Returns the time coordinate of the chart point the Expert Advisor or script has been dropped to
<u>ChartXOnDropped</u>	Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to
<u>ChartYOnDropped</u>	Returns the Y coordinate of the chart point the Expert Advisor or script has been dropped to
<u>ChartSetSymbolPeriod</u>	Changes the symbol value and a period of the specified chart

ChartScreenShot

Provides a screenshot of the chart of its current state in a GIF, PNG or BMP format depending on specified extension

ChartApplyTemplate

Applies a specific template from a specified file to the chart. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool ChartApplyTemplate(  
    long      chart_id,      // Chart ID  
    const string filename    // Template file name  
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The name of the file containing the template.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the [error](#), call the [GetLastError\(\)](#) function.

Note

The Expert Advisor will be unloaded and will not be able to continue operating in case of successful loading of a new template to the chart it is attached to.

When applying the template to the chart, trade permissions may be limited due to security reasons:

Live trading permission cannot be extended for the Expert Advisors launched by applying the template using ChartApplyTemplate() function.

If the mql5-program calling ChartApplyTemplate() function has no permission to trade, the Expert Advisor launched via the template will also not be able to trade regardless of the template settings.

If the mql5-program calling ChartApplyTemplate() function has permission to trade, while there is no such permission in the template settings, the Expert Advisor launched via the template will not be able to trade.

Using Templates

The resources of the MQL5 language allow setting multiple chart properties, including colors using the [ChartSetInteger\(\)](#) function:

- Chart background color;
- Color of the axes, scale and the OHLC line;
- Grid color;
- Color of volumes and position open levels;
- Color of the up bar, shadow and edge of a bullish candlestick;
- Color of the down bar, shadow and edge of a bearish candlestick;
- Color of the chart line and Doji candlesticks;

- Color of the bullish candlestick body;
- Color of the bearish candlestick body;
- Color of the Bid price line;
- Color of the Ask price line;
- Color of the line of the last deal price (Last);
- Color of the stop order levels (Stop Loss and Take Profit).

Besides, there can be multiple [graphical objects](#) and [indicators](#) on a chart. You may set up a chart with all the necessary indicators once and then save it as a template. Such a template can be applied to any chart.

The [ChartApplyTemplate\(\)](#) function is intended for using a previously saved template, and it can be used in any mql5 program. The path to the file that stores the template is passed as the second parameter to ChartApplyTemplate(). The template file is searched according to the following rules:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, the template is searched for relative to the path `_terminal_data_directory\MQL5`,
- if there is no backslash, the template is searched for relative to the executable EX5 file, in which ChartApplyTemplate() is called;
- if a template is not found in the first two variants, the search is performed in the folder `terminal_directory\Profiles\Templates\`.

Here `terminal_directory` is the folder from which the MetaTrader 5 Client Terminal is running, and `terminal_data_directory` is the folder, in which editable files are stored, its location depends on the operating system, user name and computer's security settings. Normally they are different folders, but in some cases they may coincide.

The location of folders `terminal_data_directory` and `terminal_directory` can be obtained using the [TerminalInfoString\(\)](#) function.

```
//--- directory from which the terminal is started
string terminal_path=TerminalInfoString(TERMINAL_PATH);
Print("Terminal directory:",terminal_path);
//--- terminal data directory, in which the MQL5 folder with EAs and indicators is located
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
Print("Terminal data directory:",terminal_data_path);
```

For example:

```
//--- search for a template in terminal_data_directory\MQL5\
ChartApplyTemplate(0,"\\first_template.tpl")
//--- search for a template in directory_of_EX5_file\, then in folder terminal_data_directory\
ChartApplyTemplate(0,"second_template.tpl")
//--- search for a template in directory_of_EX5_file\My_templates\, then in folder terminal_data_directory\
ChartApplyTemplate(0,"My_templates\\third_template.tpl")
```

Templates are not resources, they cannot be included into an executable EX5 file.

Example:

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



```
void OnStart()
{
//--- example of applying template, located in \MQL5\Files
if(FileIsExist("my_template.tpl"))
{
    Print("The file my_template.tpl found in \Files'");
    //--- apply template
    if(CharApplyTemplate(0,"\\Files\\my_template.tpl"))
    {
        Print("The template 'my_template.tpl' applied successfully");
        //--- redraw chart
        ChartRedraw();
    }
    else
        Print("Failed to apply 'my_template.tpl', error code ",GetLastError());
}
else
{
    Print("File 'my_template.tpl' not found in "
        +TerminalInfoString(TERMINAL_PATH)+"\\MQL5\\Files");
}
}
```

See also

[Resources](#)

ChartSaveTemplate

Saves current chart settings in a template with a specified name.

```
bool ChartSaveTemplate(
    long      chart_id,      // Chart ID
    const string filename    // Filename to save the template
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The filename to save the template. The ".tpl" extension will be added to the filename automatically; there is no need to specify it. The template is saved in **data_folder\Profiles\Templates** and can be used for manual application in the terminal. If a template with the same filename already exists, the contents of this file will be overwritten.

Return Value

If successful, the function returns true, otherwise it returns false. To get information about [the error](#), call the [GetLastError\(\)](#) function.

Note

Using templates, you can save chart settings with all applied indicators and graphical objects, to then apply it to another chart.

Example:

```
//+-----+
//|                                     Test_ChartSaveTemplate.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input string        symbol="GBPUSD"; // The symbol of a new chart
input ENUM_TIMEFRAMES period=PERIOD_H3; // The timeframe of a new chart
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- First attach indicators to the chart
    int handle;
    //--- Prepare the indicator for use
```



```

    if(!PrepareZigzag(NULL,0,handle)) return; // Failed, so exit
//--- Attach the indicator to the current chart, but in a separate window.
    if(!ChartIndicatorAdd(0,1,handle))
    {
        PrintFormat("Failed to attach to chart %s/%s an indicator with the handle=%d. Error code %d",
                     _Symbol,
                     EnumToString(_Period),
                     handle,
                     GetLastError());

        //--- Terminate the program operation
        return;
    }
//--- Refresh the chart to see the indicator
    ChartRedraw();
//--- Find the last two last fractures of the zigzag
    double two_values[];
    datetime two_times[];
    if(!GetLastTwoFractures(two_values,two_times,handle))
    {
        PrintFormat("Failed to find two last fractures in the Zigzag!");
        //--- Terminate the program operation
        return;
    }
//--- Now attach a standard deviation channel
    string channel="StdDeviation Channel";
    if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
    {
        PrintFormat("Failed to create object %s. Error code %d",
                     EnumToString(OBJ_STDDEVCHANNEL),GetLastError());

        return;
    }
    else
    {
        //--- The channel has been created, define the second point
        ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
        //--- Set a tooltip text for the channel
        ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL5 Help");
        //--- Refresh the chart
        ChartRedraw();
    }
//--- Save the result in a template
    ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- Open a new chart and apply a saved template to it
    long new_chart=ChartOpen(symbol,period);
    //--- Enable tooltips for graphical objects
    ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
    if(new_chart!=0)
    {
        //--- Apply the saved template to a chart

```



```

        ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
    }
    Sleep(10000);
}
//+-----+
//| Creates a zigzag handle and ensures readiness of its data |
//+-----+
bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
{
    ResetLastError();
    //--- The Zigzag indicator must be located in terminal_data_folder\MQL5\Examples
    h=iCustom(sym,tf,"Examples\\Zigzag");
    if(h==INVALID_HANDLE)
    {
        PrintFormat("%s: Failed to create the handle of the Zigzag indicator. Error code
                    __FUNCTION__,GetLastError());
        return false;
    }
    //--- When creating an indicator handle, it requires time to calculate values
    int k=0; // The number of attempts to wait for the indicator calculation
    //--- Wait for the calculation in a loop, pausing to 50 milliseconds if the calculation
    while(BarsCalculated(h)<=0)
    {
        k++;
        //--- Show the number of attempts
        PrintFormat("%s: k=%d",__FUNCTION__,k);
        //--- Wait 50 milliseconds to wait until the indicator is calculated
        Sleep(50);
        //--- If more than 100 attempt, then something is wrong
        if(k>100)
        {
            //--- Report a problem
            PrintFormat("Failed to calculate the indicator for %d attempts!");
            //--- Terminate the program operation
            return false;
        }
    }
    //--- Everything is ready, the indicator is created and values are calculated
    return true;
}
//+-----+
//| Searches for the last 2 zigzag fractures and places to arrays |
//+-----+
bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int handle)
{
    double values[]; // An array for the values of the zigzag
    datetime times[]; // An array to get time
    int size=100; // Size of the array
    ResetLastError();

```



```

//--- Copy the last 100 values of the indicator
int copied=CopyBuffer(handle,0,0,size,values);
//--- Check the number of values copied
if(copied<100)
{
    PrintFormat("%s: Failed to copy %d values of the indicator with the handle=%d. Error code %d",
        __FUNCTION__, size, handle, GetLastError());
    return false;
}
//--- Define the order of access to the array as in a timeseries
ArraySetAsSeries(values,true);
//--- Write here the numbers of bars, in which fractures were found
int positions[];
//--- Set array sizes
ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positions,3);
//--- Counters
int i=0,k=0;
//--- Start to search for fractures
while(i<100)
{
    double v=values[i];
    //--- We are not interested in empty values
    if(v!=0.0)
    {
        //--- Remember the bar number
        positions[k]=i;
        //--- Remember the value of a zigzag on the fracture
        get_values[k]=values[i];
        PrintFormat("%s: Zigzag[%d]=%G", __FUNCTION__, i, values[i]);
        //--- Increase the counter
        k++;
        //--- If two fractures found, break the loop
        if(k>2) break;
    }
    i++;
}
//--- Define the order of access to the arrays as in a timeseries
ArraySetAsSeries(times,true); ArraySetAsSeries(get_times,true);
if(CopyTime(_Symbol,_Period,0,size,times)<=0)
{
    PrintFormat("%s: Failed to copy %d values from CopyTime(). Error code %d",
        __FUNCTION__, size, GetLastError());
    return false;
}
//--- Open the bar open time, on which the last 2 fractures occurred
get_times[0]=times[positions[1]]; // The last but one value will be written as the first
get_times[1]=times[positions[2]]; // The value third from the end will be the second
PrintFormat("%s: first=%s, second=%s", __FUNCTION__, TimeToString(get_times[1]), TimeToString(get_times[0]));
//--- Successful

```



```
    return true;  
}
```

See also

[ChartApplyTemplate\(\)](#), [Resources](#)

ChartWindowFind

The function returns the number of a subwindow where an indicator is drawn. There are 2 variants of the function.

1. The function searches in the indicated chart for the subwindow with the specified "short name" of the indicator (the short name is displayed in the left top part of the subwindow), and it returns the subwindow number in case of success.

```
int ChartWindowFind(
    long      chart_id,           // chart identifier
    string     indicator_shortcode // short indicator name, see INDICATOR\_SHORTNAME
```

2. The function must be called from a custom indicator. It returns the number of the subwindow where the indicator is working.

```
int ChartWindowFind();
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

indicator_shortcode

[in] Short name of the indicator.

Return Value

Subwindow number in case of success. In case of failure the function returns -1.

Note

If the second variant of the function (without parameters) is called from a script or Expert Advisor, the function returns -1.

Don't mix up the short name of an indicator and a file name, which is specified when an indicator is created using [iCustom\(\)](#) and [IndicatorCreate\(\)](#) functions. If the indicator's short name is not set explicitly, then the name of the file containing the source code of the indicator, is specified in it during compilation.

It is important to correctly form the short name of an indicator, which is recorded in the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that the short name contains values of the indicator's input parameters, because the indicator deleted from a chart in the [ChartIndicatorDelete\(\)](#) function is identified by its short name.

Example:

```
#property script_show_inputs
//--- input parameters
input string  shortcode="MACD(12,26,9)";
//+-----+
//| Returns number of the chart window with this indicator |
//+-----+
int GetIndicatorSubWindowNumber(long chartID=0, string short_name="")
```



```

{
    int window=-1;
    //---
    if ((ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR)
    {
        //--- the function is called from the indicator, name is not required
        window=ChartWindowFind();
    }
    else
    {
        //--- the function is called from an Expert Advisor or script
        window=ChartWindowFind(0,short_name);
        if(window==-1) Print(__FUNCTION__+"(): Error = ",GetLastError());
    }
    //---
    return(window);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //---
    int window=GetIndicatorSubWindowNumber(0,shortname);
    if(window!=-1)
        Print("Indicator "+shortname+" is in the window #"+(string)window);
    else
        Print("Indicator "+shortname+" is not found. window = "+(string)window);
}

```

See also

[ObjectCreate\(\)](#), [ObjectFind\(\)](#)

ChartTimePriceToXY

Converts the coordinates of a chart from the time/price representation to the X and Y coordinates.

```
bool ChartTimePriceToXY(
    long      chart_id,    // Chart ID
    int       sub_window,  // The number of the subwindow
    datetime  time,        // Time on the chart
    double    price,       // Price on the chart
    int&      x,           // The X coordinate for the time on the chart
    int&      y            // The Y coordinates for the price on the chart
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart subwindow. 0 means the main chart window.

time

[in] The time value on the chart, for which the value in pixels along the X axis will be received. The origin is in the upper left corner of the main chart window.

price

[in] The price value on the chart, for which the value in pixels along the Y axis will be received. The origin is in the upper left corner of the main chart window.

x

[out] The variable, into which the conversion of time to X will be received.

y

[out] The variable, into which the conversion of price to Y will be received.

Return Value

Returns true if successful, otherwise false. To get information about [the error](#), call the [GetLastError\(\)](#) function.

See also

[ChartXYToTimePrice\(\)](#)

ChartXYToTimePrice

Converts the X and Y coordinates on a chart to the time and price values.

```
bool ChartXYToTimePrice(
    long      chart_id,    // Chart ID
    int       x,           // The X coordinate on the chart
    int       y,           // The Y coordinate on the chart
    int&      sub_window,  // The number of the subwindow
    datetime& time,        // Time on the chart
    double&   price       // Price on the chart
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

x

[in] The X coordinate.

y

[in] The Y coordinate.

sub_window

[out] The variable, into which the chart subwindow number will be written. 0 means the main chart window.

time

[out] The time value on the chart, for which the value in pixels along the X axis will be received. The origin is in the upper left corner of the main chart window.

price

[out] The price value on the chart, for which the value in pixels along the Y axis will be received. The origin is in the upper left corner of the main chart window.

Return Value

Returns true if successful, otherwise false. To get information about [the error](#), call the [GetLastError\(\)](#) function.

Example:

```
//+-----+
//| ChartEvent function |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
    //-- Show the event parameters on the chart
```



```

Comment(__FUNCTION__,": id=",id," lparam=",lparam," dparam=",dparam," sparam=",sparam);
//--- If this is an event of a mouse click on the chart
if(id==CHARTEVENT_CLICK)
{
    //--- Prepare variables
    int      x      =(int)lparam;
    int      y      =(int)dparam;
    datetime dt      =0;
    double   price   =0;
    int      window  =0;
    //--- Convert the X and Y coordinates in terms of date/time
    if(CharTimePriceToXY(0,x,y,window,dt,price))
    {
        PrintFormat("Window=%d X=%d Y=%d => Time=%s Price=%G",window,x,y,TimeToString(dt),price);
        //--- Perform reverse conversion: (X,Y) => (Time,Price)
        if(CharTimePriceToXY(0,window,dt,price,x,y))
            PrintFormat("Time=%s Price=%G => X=%d Y=%d",TimeToString(dt),price,x,y);
        else
            Print("ChartTimePriceToXY return error code: ",GetLastError());
        //--- delete lines
        ObjectDelete(0,"V Line");
        ObjectDelete(0,"H Line");
        //--- create horizontal and vertical lines of the crosshair
        ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
        ObjectCreate(0,"V Line",OBJ_VLINE,window,dt,price);
        ChartRedraw(0);
    }
    else
        Print("ChartTimePriceToXY return error code: ",GetLastError());
    Print("+-----+");
}
}

```

See also

[ChartTimePriceToXY\(\)](#)

ChartOpen

Opens a new chart with the specified symbol and period.

```
long ChartOpen(  
    string      symbol,      // Symbol name  
    ENUM_TIMEFRAMES period  // Period  
);
```

Parameters

symbol

[in] Chart symbol. [NULL](#) means the symbol of the current chart (the Expert Advisor is attached to).

period

[in] Chart period (timeframe). Can be one of the [ENUM_TIMEFRAMES](#) values. 0 means the current chart period.

Return Value

If successful, it returns the opened chart ID. Otherwise returns 0.

Note

The maximum possible number of simultaneously open charts in the terminal can't exceed the [CHARTS_MAX](#) value.

ChartFirst

Returns the ID of the first chart of the client terminal.

```
long ChartFirst();
```

Return Value

Chart ID.

ChartNext

Returns the chart ID of the chart next to the specified one.

```
long ChartNext(  
    long chart_id    // Chart ID  
);
```

Parameters

chart_id

[in] Chart ID. 0 does not mean the current chart. 0 means "return the first chart ID".

Return Value

Chart ID. If this is the end of the chart list, it returns -1.

Example:

```
//--- variables for chart ID  
long currChart,prevChart=ChartFirst();  
int i=0,limit=100;  
Print("ChartFirst =",ChartSymbol(prevChart)," ID =",prevChart);  
while(i<limit)// We have certainly not more than 100 open charts  
{  
    currChart=ChartNext(prevChart); // Get the new chart ID by using the previous chart ID  
    if(currChart<0) break;           // Have reached the end of the chart list  
    Print(i,ChartSymbol(currChart)," ID =",currChart);  
    prevChart=currChart;// let's save the current chart ID for the ChartNext()  
    i++;// Do not forget to increase the counter  
}
```


ChartClose

Closes the specified chart.

```
bool ChartClose(  
    long chart_id=0    // Chart ID  
);
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If successful, returns true, otherwise false.

ChartSymbol

Returns the symbol name for the specified chart.

```
string ChartSymbol(  
    long chart_id=0    // Chart ID  
);
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If chart does not exist, the result will be an empty string.

See also

[ChartSetSymbolPeriod](#)

ChartPeriod

Returns the timeframe [period](#) of specified chart.

```
ENUM_TIMEFRAMES ChartPeriod(  
    long chart_id=0    // Chart ID  
);
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

The function returns one of the [ENUM_TIMEFRAMES](#) values. If chart does not exist, it returns 0.

ChartRedraw

This function calls a forced redrawing of a specified chart.

```
void ChartRedraw(  
    long chart_id=0    // Chart ID  
);
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Note

Usually it is used after changing the [object properties](#).

See also

[Objects functions](#)

ChartSetDouble

Sets a value for a corresponding property of the specified chart. Chart property should be of a [double](#) type. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool ChartSetDouble(  
    long    chart_id,    // Chart ID  
    int     prop_id,     // Property ID  
    double  value        // Value  
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Can be one of the [ENUM_CHART_PROPERTY_DOUBLE](#) values (except the read-only properties).

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the [error](#), call the [GetLastError\(\)](#) function.

ChartSetInteger

Sets a value for a corresponding property of the specified chart. Chart property must be [datetime](#), [int](#), [color](#), [bool](#) or [char](#). The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool ChartSetInteger(  
    long   chart_id,      // Chart ID  
    int    prop_id,      // Property ID  
    long   value          // Value  
);
```

Sets a value for a corresponding property of the specified subwindow.

```
bool ChartSetInteger(  
    long   chart_id,      // Chart ID  
    int    prop_id,      // Property ID  
    int    sub_window,    // Subwindow number  
    long   value          // Value  
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. It can be one of the [ENUM_CHART_PROPERTY_INTEGER](#) value (except the read-only properties).

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the [error](#), call the [GetLastError\(\)](#) function.

ChartSetString

Sets a value for a corresponding property of the specified chart. Chart property must be of the string type. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool ChartSetString(
    long  chart_id,      // Chart ID
    int   prop_id,       // Property ID
    string str_value     // Value
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Its value can be one of the [ENUM_CHART_PROPERTY_STRING](#) values (except the read-only properties).

str_value

[in] Property value string. String length cannot exceed 2045 characters (extra characters will be truncated).

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the [error](#), call the [GetLastError\(\)](#) function.

Note

ChartSetString can be used for a comment output on the chart instead of the [Comment](#) function.

Example:

```
void OnTick()
{
    //---
    double Ask,Bid;
    int Spread;
    Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
    Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
    string comment=StringFormat("Printing prices:\nAsk = %G\nBid = %G\nSpread = %d",
                                Ask,Bid,Spread);
    ChartSetString(0,CHART_COMMENT,comment);
}
```

See also

[Comment](#), [ChartGetString](#)

ChartGetDouble

Returns the value of a corresponding property of the specified chart. Chart property must be of double type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
double ChartGetDouble(  
    long  chart_id,          // Chart ID  
    int   prop_id,          // Property ID  
    int   sub_window=0      // subwindow number, if necessary  
);
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable double_var passed by reference.

```
bool ChartGetDouble(  
    long  chart_id,          // Chart ID  
    int   prop_id,          // Property ID  
    int   sub_window,       // Subwindow number  
    double& double_var      // Target variable for the chart property  
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the [ENUM_CHART_PROPERTY_DOUBLE](#) values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

double_var

[out] Target variable of double type for the requested property.

Return Value

The value of double type.

For the second call case it returns true if the specified property is available and its value has been placed into double_var variable, otherwise returns false. To get an additional information about the [error](#), it is necessary to call the function [GetLastError\(\)](#).

Example:

```
void OnStart()  
{  
    double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,0);  
    double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,0);  
    Print("CHART_PRICE_MIN =",priceMin);  
}
```



```
Print("CHART_PRICE_MAX =",priceMax);  
}
```


ChartGetInteger

Returns the value of a corresponding property of the specified chart. Chart property must be of [datetime](#), [int](#) or [bool](#) type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
long ChartGetInteger(
    long  chart_id,      // Chart ID
    int   prop_id,       // Property ID
    int   sub_window=0   // subwindow number, if necessary
);
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable long_var passed by reference.

```
bool ChartGetInteger(
    long  chart_id,      // Chart ID
    int   prop_id,       // Property ID
    int   sub_window,    // subwindow number
    long& long_var       // Target variable for the property
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the [ENUM_CHART_PROPERTY_INTEGER](#) values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

long_var

[out] Target variable of long type for the requested property.

Return Value

The value of long type.

For the second call case it returns true if specified property is available and its value has been stored into long_var variable, otherwise returns false. To get additional information about the [error](#), it is necessary to call the function [GetLastError\(\)](#).

Example:

```
void OnStart()
{
    int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
    int width=ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
    Print("CHART_HEIGHT_IN_PIXELS =",height,"pixels");
}
```



```
Print("CHART_WIDTH_IN_PIXELS =",width,"pixels");  
}
```


ChartGetString

Returns the value of a corresponding property of the specified chart. Chart property must be of string type. There are 2 variants of the function call.

1. Returns the property value directly.

```
string ChartGetString(
    long  chart_id,      // Chart ID
    int   prop_id        // Property ID
);
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable string_var passed by reference.

```
bool ChartGetString(
    long  chart_id,      // Chart ID
    int   prop_id,      // Property ID
    string& string_var   // Target variable for the property
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the [ENUM_CHART_PROPERTY_STRING](#) values.

string_var

[out] Target variable of string type for the requested property.

Return Value

The value of string type.

For the second call case it returns true if the specified property is available and its value has been stored into string_var variable, otherwise returns false. To get additional information about the [error](#), it is necessary to call the function [GetLastError\(\)](#).

Note

ChartGetString can be used for reading comments plotted on the chart using the [Comment](#) or [ChartSetString](#) functions.

Example:

```
void OnStart()
{
    ChartSetString(0, CHART_COMMENT, "Test comment.\nSecond line.\nThird!");
    ChartRedraw();
    Sleep(1000);
    string comm=ChartGetString(0, CHART_COMMENT);
    Print(comm);
}
```



```
}
```

See also

[Comment](#), [ChartSetString](#)

ChartNavigate

Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart.

```
bool ChartNavigate(
    long  chart_id,    // Chart ID
    int   position,    // Position
    int   shift=0      // Shift value
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

position

[in] Chart position to perform a shift. Can be one of the [ENUM_CHART_POSITION](#) values.

shift=0

[in] Number of bars to shift the chart. Positive value means the right shift (to the end of chart), negative value means the left shift (to the beginning of chart). The zero shift can be used to navigate to the beginning or end of chart.

Return Value

Returns true if successful, otherwise returns false.

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- get handle of the current chart
    long handle=ChartID();
    string comm="";
    if(handle>0) // if successful, additionally set up the chart
    {
        //--- disable auto scroll
        ChartSetInteger(handle,CHART_AUTOSCROLL,false);
        //--- set a shift from the right chart border
        ChartSetInteger(handle,CHART_SHIFT,true);
        //--- draw candlesticks
        ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
        //--- set the display mode for tick volumes
        ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

        //--- prepare a text to output in Comment()
        comm="Scroll 10 bars to the right of the history start";
        //--- show comment
```



```

Comment(comm);
//--- scroll 10 bars to the right of the history start
ChartNavigate(handle,CHART_BEGIN,10);
//--- get the number of the first bar visible on the chart (numeration like in t
long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
//--- add line feed character
comm=comm+"\r\n";
//--- add to comment
comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\n";
//--- show comment
Comment(comm);
//--- wait 5 seconds to see how the chart moves
Sleep(5000);

//--- add to the comment text
comm=comm+"\r\n"+"Scroll 10 bars to the left of the right chart border";
Comment(comm);
//--- scroll 10 bars to the left of the right chart border
ChartNavigate(handle,CHART_END,-10);
//--- get the number of the first bar visible on the chart (numeration like in t
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\n";
Comment(comm);
//--- wait 5 seconds to see how the chart moves
Sleep(5000);

//--- new block of chart scrolling
comm=comm+"\r\n"+"Scroll 300 bars to the right of the history start";
Comment(comm);
//--- scroll 300 bars to the right of the history start
ChartNavigate(handle,CHART_BEGIN,300);
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\n";
Comment(comm);
//--- wait 5 seconds to see how the chart moves
Sleep(5000);

//--- new block of chart scrolling
comm=comm+"\r\n"+"Scroll 300 bars to the left of the right chart border";
Comment(comm);
//--- scroll 300 bars to the left of the right chart border
ChartNavigate(handle,CHART_END,-300);
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\n";
Comment(comm);
}

```



```
}
```


ChartID

Returns the ID of the current chart.

```
long ChartID();
```

Return Value

Value of [long](#) type.

ChartIndicatorAdd

Adds an indicator with the specified handle into a specified chart window. Indicator and chart should be generated on the same symbol and time frame.

```
bool ChartIndicatorAdd(
    long   chart_id,           // chart ID
    int    sub_window         // number of the sub-window
    int    indicator_handle    // handle of the indicator
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart sub-window. 0 means the main chart window. To add an indicator in a new window, the parameter must be one greater than the index of the last existing window, i.e. equal to [CHART_WINDOWS_TOTAL](#). If the value of the parameter is greater than [CHART_WINDOWS_TOTAL](#), a new window will not be created, and the indicator will not be added.

indicator_handle

[in] The handle of the indicator.

Return Value

The function returns true in case of success, otherwise it returns false. In order to obtain information about the [error](#), call the [GetLastError\(\)](#) function. Error 4114 means that a chart and an added indicator differ by their symbol or time frame.

Note

If an indicator that should be drawn in a separate subwindow (for example, built-in [iMACD](#) or a custom indicator with specified [#property indicator_separate_window](#) property) is applied to the main chart window, it may not be visible though it will still be present in the list of indicators. This means that the scale of the indicator is different from the scale of the price chart, and applied indicator's values do not fit in the displayed range of the price chart. In this case, [GetLastError\(\)](#) returns zero code indicating the absence of an error. The values of such "invisible" indicator can be seen in Data Window and received from other MQL5 applications.

Example:

```
#property description "Expert Advisor demonstrating the work with ChartIndicatorAdd()
#property description "After launching on the chart (and receiving the error in Journal)
#property description "the Expert Advisor's properties and specify correct <symbol> and
#property description "MACD indicator will be added on the chart."

//--- input parameters
input string      symbol="AUDUSD";      // symbol name
input ENUM_TIMEFRAMES period=PERIOD_M12; // time frame
input int         fast_ema_period=12;    // fast MACD period
input int         slow_ema_period=26;    // slow MACD period
```



```

input int      signal_period=9;           // signal period
input ENUM_APPLIED_PRICE apr=PRICE_CLOSE; // price type for MACD calculation

int indicator_handle=INVALID_HANDLE;

//+-----+
//| Expert initialization function          |
//+-----+
int OnInit()
{
//---
    indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_period,
//--- try to add the indicator on the chart
    if(!AddIndicator())
    {
        //--- AddIndicator() function refused to add the indicator on the chart
        int answer=MessageBox("Do you want to add MACD on the chart anyway?",
                                "Incorrect symbol and/or time frame for adding the indicator",
                                MB_YESNO // "Yes" and "No" selection buttons will be shown
                                );
        //--- if a user still insists on incorrect usage of ChartIndicatorAdd()
        if(answer==IDYES)
        {
            //--- first of all, a Journal entry will be made about that
            PrintFormat("Attention! %s: Trying to add MACD(%s/%s) indicator on %s/%s chart window. Error code %d",
                        __FUNCTION__,symbol,EnumToString(period),_Symbol,EnumToString(_Period),
                        GetLastError());
            //--- receive the number of a new subwindow, to which we will try to add the indicator
            int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
            //--- now make an attempt resulting in error
            if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
                PrintFormat("Failed to add MACD indicator on %d chart window. Error code %d",
                            subwindow,GetLastError());
        }
    }
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert tick function                  |
//+-----+
void OnTick()
{
    // Expert Advisor performs nothing
}

//+-----+
//| Function for checking and adding the indicator on the chart          |
//+-----+
bool AddIndicator()
{
    //--- displayed message

```



```

    string message;
    //--- check if the indicator symbol and chart symbol match each other
    if(symbol!=_Symbol)
    {
        message="Displaying the use of Demo_ChartIndicatorAdd() function:";
        message=message+"\r\n";
        message=message+"Unable to add the indicator calculated on another symbol on the chart:";
        message=message+"\r\n";
        message=message+"Specify the chart symbol in Expert Advisor's property - "+_Symbol;
        Alert(message);
        //--- premature exit, the indicator will not be added on the chart
        return false;
    }
    //--- check if the indicator's and chart's time frames match each other
    if(period!=_Period)
    {
        message="Unable to add the indicator calculated on another time frame on the chart:";
        message=message+"\r\n";
        message=message+"Specify the chart time frame in Expert Advisor properties - "+_Period;
        Alert(message);
        //--- premature exit, the indicator will not be added on the chart
        return false;
    }
    //--- all checks completed, symbol and indicator time frame match the chart
    if(indicator_handle==INVALID_HANDLE)
    {
        Print(__FUNCTION__," Creating MACD indicator");
        indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_period);
        if(indicator_handle==INVALID_HANDLE)
        {
            Print("Failed to create MACD indicator. Error code ",GetLastError());
        }
    }
    //--- reset the error code
    ResetLastError();
    //--- apply the indicator to the chart
    Print(__FUNCTION__," Adding MACD indicator on the chart");
    Print("MACD is generated on ",symbol,"/",EnumToString(period));
    //--- receive the number of a new subwindow, to which MACD indicator is added
    int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
    PrintFormat("Adding MACD indicator on %d chart window",subwindow);
    if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
    {
        PrintFormat("Failed to add MACD indicator on %d chart window. Error code %d",
            subwindow,GetLastError());
    }
    //--- Indicator added successfully
    return(true);
}

```


See Also

[ChartIndicatorDelete\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#)

ChartIndicatorDelete

Removes an indicator with a specified name from the specified chart window.

```
bool ChartIndicatorDelete(
    long      chart_id,           // chart id
    int       sub_window         // number of the subwindow
    const string indicator_shortcode // short name of the indicator
);
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const indicator_shortcode

[in] The short name of the indicator which is set in the [INDICATOR_SHORTNAME](#) property with the [IndicatorSetString\(\)](#) function. To get the short name of an indicator use the [ChartIndicatorName\(\)](#) function.

Return Value

Returns true in case of successful deletion of the indicator. Otherwise it returns false. To get [error](#) details use the [GetLastError\(\)](#) function.

Note

If two indicators with identical short names exist in the chart subwindow, the first one in a row will be deleted.

If other indicators on this chart are based on the values of the indicator that is being deleted, such indicators will also be deleted.

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions [iCustom\(\)](#) and [IndicatorCreate\(\)](#). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

Deletion of an indicator from a chart doesn't mean that its calculation part will be deleted from the terminal memory. To release the indicator handle use the [IndicatorRelease\(\)](#) function.

The indicator's short name should be formed correctly. It will be written to the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the [ChartIndicatorDelete\(\)](#) function is identified by the short name.

Example of deleting an indicator after initialization has failed:

```
//+-----+
//|                                     Demo_ChartIndicatorDelete.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
```



```

//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots  1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      first_param=1;
input int      second_param=2;
input int      third_param=3;
input bool     wrong_init=true;
//--- indicator buffers
double         HistogramBuffer[];
string         shortname;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    int res=INIT_SUCCEEDED;
//--- Link the HistogramBuffer array to the indicator buffer
    SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- Construct a short indicator name based on input parameters
    shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                           first_param,second_param,third_param);
    IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- If forced completion of an indicator is set, return a non-zero value
    if(wrong_init) res=INIT_FAILED;
    return(res);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],

```



```

        const long &volume[],
        const int &spread[])
    {
        //--- Starting position for working in a loop
        int start=prev_calculated-1;
        if(start<0) start=0;
        //--- Fill in the indicator buffer with values
        for(int i=start;i<rates_total;i++)
        {
            HistogramBuffer[i]=close[i];
        }
        //--- return value of prev_calculated for next call
        return(rates_total);
    }
    //+-----+
    //| A handler of the Deinit event |
    //+-----+
    void OnDeinit(const int reason)
    {
        PrintFormat("%s: Deinitialization reason code=%d",__FUNCTION__,reason);
        if(reason==REASON_INITFAILED)
        {
            PrintFormat("An indicator with a short name %s (file %s) deletes itself from the
            int window=ChartWindowFind();
            bool res=ChartIndicatorDelete(0,window,shortname);
            //--- Analyse the result of call of ChartIndicatorDelete()
            if(!res)
            {
                PrintFormat("Failed to delete indicator %s from window #%. Error code %d",
                    shortname,window,GetLastError());
            }
        }
    }
}

```

See also

[ChartIndicatorAdd\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

ChartIndicatorGet

Returns the handle of the indicator with the specified short name in the specified chart window.

```
int ChartIndicatorGet(
    long      chart_id,           // Chart ID
    int       sub_window         // The number of the subwindow
    const string indicator_shortcode // Short name of the indicator
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart subwindow. 0 means the main chart window.

const indicator_shortcode

[in] The short name of the indicator, which is set in the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. To get the short name of an indicator, use the [ChartIndicatorName\(\)](#) function.

Return Value

Returns an indicator handle if successful, otherwise returns [INVALID_HANDLE](#). To get information about [the error](#), call the [GetLastError\(\)](#) function.

Note

When creating an indicator, be careful forming its short name, which is written in the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that a short name should contain the values of input parameters of the indicator, since the indicator is identified in the [ChartIndicatorGet\(\)](#) function based on its short name.

Another way to identify the indicator is to get a list of its parameters for a given handle using the [IndicatorParameters\(\)](#) function and then to analyze the obtained values.

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- The number of windows on the chart (at least one main window is always present)
    int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
    //--- Check all windows
    for(int w=0;w<windows;w++)
    {
        //--- the number of indicators in this window/subwindow
        int total=ChartIndicatorsTotal(0,w);
        //--- Go through all indicators in the window
```



```
for(int i=0;i<total;i++)
{
    ///--- get the short name of an indicator
    string name=ChartIndicatorName(0,w,i);
    ///--- get the handle of an indicator
    int handle=ChartIndicatorGet(0,w,name);
    ///--- Add to log
    PrintFormat("Window=%d, index=%d, name=%s, handle=%d",w,i,name,handle);
    ///--- You should obligatorily release the indicator handle when it is no longer needed
    IndicatorRelease(handle);
}
}
```

See also

[ChartIndicatorAdd\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [IndicatorParameters\(\)](#)

ChartIndicatorName

Returns the short name of the indicator by the number in the indicators list on the specified chart window.

```
string ChartIndicatorName (
    long   chart_id,      // chart id
    int    sub_window     // number of the subwindow
    int    index          // index of the indicator in the list of indicators added to t
);
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

index

[in] the index of the indicator in the list of indicators. The numeration of indicators start with zero, i.e. the first indicator in the list has the 0 index. To obtain the number of indicators in the list use the [ChartIndicatorsTotal\(\)](#) function.

Return Value

The short name of the indicator which is set in the [INDICATOR_SHORTNAME](#) property with the [IndicatorSetString\(\)](#) function. To get [error](#) details use the [GetLastError\(\)](#) function.

Note

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions [iCustom\(\)](#) and [IndicatorCreate\(\)](#). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the [ChartIndicatorDelete\(\)](#) function is identified by the short name.

See also

[ChartIndicatorAdd\(\)](#), [ChartIndicatorDelete\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

ChartIndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int ChartIndicatorsTotal(  
    long chart_id,      // chart id  
    int sub_window      // number of the subwindow  
);
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

Return Value

The number of indicators in the specified chart window. To get [error](#) details use the [GetLastError\(\)](#) function.

Note

The function allows going searching through all the indicators attached to the chart. The number of all the windows of the chart can be obtained from the [CHART_WINDOWS_TOTAL](#) property using the [ChartGetInteger\(\)](#) function.

See also

[ChartIndicatorAdd\(\)](#), [ChartIndicatorDelete\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

ChartWindowOnDropped

Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to. 0 means the main chart window.

```
int ChartWindowOnDropped();
```

Return Value

Value of [int](#) type.

Example:

```
int myWindow=ChartWindowOnDropped();  
int windowsTotal=ChartGetInteger(0,CHART_WINDOWS_TOTAL);  
Print("Script is running on the window #"+myWindow+  
      ". Total windows on the chart "+ChartSymbol()+":",windowsTotal);
```

See also

[ChartPriceOnDropped](#), [ChartTimeOnDropped](#), [ChartXOnDropped](#), [ChartYOnDropped](#)

ChartPriceOnDropped

Returns the price coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
double ChartPriceOnDropped();
```

Return Value

Value of [double](#) type.

Example:

```
double p=ChartPriceOnDropped();  
Print("ChartPriceOnDropped() = ",p);
```

See also

[ChartXOnDropped](#), [ChartYOnDropped](#)

ChartTimeOnDropped

Returns the time coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
datetime ChartTimeOnDropped();
```

Return Value

Value of [datetime](#) type.

Example:

```
datetime t=ChartTimeOnDropped();  
Print("Script was dropped on the "+t);
```

See also

[ChartXOnDropped](#), [ChartYOnDropped](#)

ChartXOnDropped

Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to.

```
int ChartXOnDropped();
```

Return Value

The X coordinate value.

Note

X axis direction from left to right.

Example:

```
int X=ChartXOnDropped();  
int Y=ChartYOnDropped();  
Print(" (X,Y) = (" +X+" , "+Y+" ) ");
```

See also

[ChartWindowOnDropped](#), [ChartPriceOnDropped](#), [ChartTimeOnDropped](#)

ChartYOnDropped

Returns the Y coordinate of the chart point the Expert Advisor or script has been dropped to.

```
int ChartYOnDropped();
```

Return Value

The Y coordinate value.

Note

Y axis direction from top to bottom.

See also

[ChartWindowOnDropped](#), [ChartPriceOnDropped](#), [ChartTimeOnDropped](#)

ChartSetSymbolPeriod

Changes the symbol and period of the specified chart. The function is asynchronous, i.e. it sends the command and does not wait for its execution completion. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool ChartSetSymbolPeriod(  
    long          chart_id,    // Chart ID  
    string        symbol,      // Symbol name  
    ENUM_TIMEFRAMES period    // Period  
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

symbol

[in] Chart symbol. [NULL](#) value means the current chart symbol (Expert Advisor is attached to)

period

[in] Chart period (timeframe). Can be one of the [ENUM_TIMEFRAMES](#) values. 0 means the current chart period.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the [error](#), call the [GetLastError\(\)](#) function.

Note

The symbol/period change leads to the re-initialization of the Expert Advisor attached to a chart.

The call of ChartSetSymbolPeriod with the same symbol and timeframe can be used to update the chart (similar to the terminal's Refresh command). In its turn, the chart update triggers recalculation of the indicators attached to it. Thus, you are able to calculate an indicator on the chart even if there are no ticks (e.g., on weekends).

See also

[ChartSymbol](#), [ChartPeriod](#)

ChartScreenShot

The function provides a screenshot of the chart in its current state in the GIF, PNG or BMP format depending on specified extension.

```
bool ChartScreenShot(
    long      chart_id,           // Chart ID
    string     filename,          // Symbol name
    int        width,             // Width
    int        height,            // Height
    ENUM_ALIGN_MODE align_mode=ALIGN_RIGHT // Alignment type
);
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] Screenshot file name. Cannot exceed 63 characters. Screenshot files are placed in the \Files directory.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Output mode of a narrow screenshot. A value of the [ENUM_ALIGN_MODE](#) enumeration. ALIGN_RIGHT means align to the right margin (the output from the end). ALIGN_LEFT means Left justify.

Return Value

Returns true if successful, otherwise false.

Note

If you need to take a screenshot from a chart from a certain position, first it's necessary to position the graph using the [ChartNavigate\(\)](#) function. If the horizontal size of the screenshot is smaller than the chart window, either the right part of the chart window, or its left part is output, depending on the align_mode settings.

Example:

```
#property description "The Expert Advisor demonstrates how to create a series of screenshots
#property description "chart using the ChartScreenShot() function. For convenience, the
#property description "shown on the chart. The height and width of images is defined to

#define WIDTH 800 // Image width to call ChartScreenShot()
#define HEIGHT 600 // Image height to call ChartScreenShot()
```



```

//--- input parameters
input int      pictures=5;    // The number of images in the series
int          mode=-1;        // -1 denotes a shift to the right edge of the chart, 1
int          bars_shift=300; // The number of bars when scrolling the chart using Cha
//+-----+
//| Expert initialization function                                     |
//+-----+
void OnInit()
{
//--- Disable chart autoscroll
    ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- Set the shift of the right edge of the chart
    ChartSetInteger(0,CHART_SHIFT,true);
//--- Show a candlestick chart
    ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
    Print("Preparation of the Expert Advisor is completed");
}
//+-----+
//| Expert tick function                                           |
//+-----+
void OnTick()
{
//---

}
//+-----+
//| ChartEvent function                                           |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- Show the name of the function, call time and event identifier
    Print(__FUNCTION__,TimeCurrent()," id=",id," mode=",mode);
//--- Handle the CHARTEVENT_CLICK event ("A mouse click on the chart")
    if(id==CHARTEVENT_CLICK)
    {
        //--- Initial shift from the chart edge
        int pos=0;
        //--- Operation with the left chart edge
        if(mode>0)
        {
            //--- Scroll the chart to the left edge
            ChartNavigate(0,CHART_BEGIN,pos);
            for(int i=0;i<pictures;i++)
            {
                //--- Prepare a text to show on the chart and a file name

```



```

        string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif";
        ///--- Show the name on the chart as a comment
        Comment(name);
        ///--- Save the chart screenshot in a file in the terminal_directory\MQL5\
        if (ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
            Print("We've saved the screenshot ",name);
        ///---
        pos+=bars_shift;
        ///--- Give the user time to look at the new part of the chart
        Sleep(3000);
        ///--- Scroll the chart from the current position bars_shift bars to the r
        ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
    }
    ///--- Change the mode to the opposite
    mode*=-1;
}
else // Operation with the right chart edge
{
    ///--- Scroll the chart to the right edge
    ChartNavigate(0,CHART_END,pos);
    for(int i=0;i<pictures;i++)
    {
        ///--- Prepare a text to show on the chart and a file name
        string name="ChartScreenShot"+"CHART_END"+string(pos)+".gif";
        ///--- Show the name on the chart as a comment
        Comment(name);
        ///--- Save the chart screenshot in a file in the terminal_directory\MQL5\
        if (ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_RIGHT))
            Print("We've saved the screenshot ",name);
        ///---
        pos+=bars_shift;
        ///--- Give the user time to look at the new part of the chart
        Sleep(3000);
        ///--- Scroll the chart from the current position bars_shift bars to the r
        ChartNavigate(0,CHART_CURRENT_POS,-bars_shift);
    }
    ///--- Change the mode to the opposite
    mode*=-1;
}
} // End of CHARTEVENT_CLICK event handling
///--- End of the OnChartEvent() handler
}

```

See also

[ChartNavigate\(\)](#), [Resources](#)

Trade Functions

This is the group of functions intended for managing trading activities.

Trading functions can be used in Expert Advisors and scripts. Trading functions can be called only if in the properties of the Expert Advisor or script the "Allow live trading" checkbox is enabled.

Trading can be allowed or prohibited depending on various factors described in the [Trade Permission](#) section.

Function	Action
OrderCalcMargin	Calculates the margin required for the specified order type, in the deposit currency
OrderCalcProfit	Calculates the profit based on the parameters passed, in the deposit currency
OrderCheck	Checks if there are enough funds to execute the required trade operation .
OrderSend	Sends trade requests to a server
OrderSendAsync	Asynchronously sends trade requests without waiting for the trade response of the trade server
PositionsTotal	Returns the number of open positions
PositionGetSymbol	Returns the symbol corresponding to the open position
PositionSelect	Chooses an open position for further working with it
PositionSelectByTicket	Selects a position to work with by the ticket number specified in it
PositionGetDouble	Returns the requested property of an open position (double)
PositionGetInteger	Returns the requested property of an open position (datetime or int)
PositionGetString	Returns the requested property of an open position (string)
PositionGetTicket	Returns the ticket of the position with the specified index in the list of open positions
OrdersTotal	Returns the number of orders
OrderGetTicket	Return the ticket of a corresponding order
OrderSelect	Selects a order for further working with it

<u>OrderGetDouble</u>	Returns the requested property of the order (double)
<u>OrderGetInteger</u>	Returns the requested property of the order (datetime or int)
<u>OrderGetString</u>	Returns the requested property of the order (string)
<u>HistorySelect</u>	Retrieves the history of transactions and orders for the specified period of the server time
<u>HistorySelectByPosition</u>	Requests the history of deals with a specified <u>position identifier</u> .
<u>HistoryOrderSelect</u>	Selects an order in the history for further working with it
<u>HistoryOrdersTotal</u>	Returns the number of orders in the history
<u>HistoryOrderGetTicket</u>	Return order ticket of a corresponding order in the history
<u>HistoryOrderGetDouble</u>	Returns the requested property of an order in the history (double)
<u>HistoryOrderGetInteger</u>	Returns the requested property of an order in the history (datetime or int)
<u>HistoryOrderGetString</u>	Returns the requested property of an order in the history (string)
<u>HistoryDealSelect</u>	Selects a deal in the history for further calling it through appropriate functions
<u>HistoryDealsTotal</u>	Returns the number of deals in the history
<u>HistoryDealGetTicket</u>	Returns a ticket of a corresponding deal in the history
<u>HistoryDealGetDouble</u>	Returns the requested property of a deal in the history (double)
<u>HistoryDealGetInteger</u>	Returns the requested property of a deal in the history (datetime or int)
<u>HistoryDealGetString</u>	Returns the requested property of a deal in the history (string)

OrderCalcMargin

The function calculates the margin required for the specified order type, on the current account, in the current market environment not taking into account current pending orders and open positions. It allows the evaluation of margin for the trade operation planned. The value is returned in the account currency.

```
bool OrderCalcMargin (
    ENUM_ORDER_TYPE   action,           // type of order
    string             symbol,          // symbol name
    double             volume,          // volume
    double             price,           // open price
    double&            margin           // variable for obtaining the margin value
);
```

Parameters

action

[in] The order type, can be one of the values of the [ENUM_ORDER_TYPE](#) enumeration.

symbol

[in] Symbol name.

volume

[in] Volume of the trade operation.

price

[in] Open price.

margin

[out] The variable, to which the value of the required margin will be written in case the function is successfully executed. The calculation is performed as if there were no pending orders and open positions on the current account. The margin value depends on many factors, and can differ in different market environments.

Return Value

The function returns true in case of success; otherwise it returns false. In order to obtain information about the [error](#), call the [GetLastError\(\)](#) function.

See also

[OrderSend\(\)](#), [Order Properties](#), [Trade Operation Types](#)

OrderCalcProfit

The function calculates the profit for the current account, in the current market conditions, based on the parameters passed. The function is used for pre-evaluation of the result of a trade operation. The value is returned in the account currency.

```
bool OrderCalcProfit (
    ENUM_ORDER_TYPE    action,           // type of the order (ORDER_TYPE_BUY or ORI
    string              symbol,          // symbol name
    double              volume,          // volume
    double              price_open,      // open price
    double              price_close,     // close price
    double&             profit           // variable for obtaining the profit value
);
```

Parameters

action

[in] Type of the order, can be one of the two values of the [ENUM_ORDER_TYPE](#) enumeration: ORDER_TYPE_BUY or ORDER_TYPE_SELL.

symbol

[in] Symbol name.

volume

[in] Volume of the trade operation.

price_open

[in] Open price.

price_close

[in] Close price.

profit

[out] The variable, to which the calculated value of the profit will be written in case the function is successfully executed. The estimated profit value depends on many factors, and can differ in different market environments.

Return Value

The function returns true in case of success; otherwise it returns false. If an invalid order type is specified, the function will return false. In order to obtain information about the [error](#), call [GetLastError\(\)](#).

See also

[OrderSend\(\)](#), [Order Properties](#), [Trade Operation Types](#)

OrderCheck

The `OrderCheck()` function checks if there are enough money to execute a required [trade operation](#). The check results are placed to the fields of the [MqlTradeCheckResult](#) structure.

```
bool OrderCheck(  
    MqlTradeRequest&    request,    // request structure  
    MqlTradeCheckResult& result    // result structure  
);
```

Parameters

request

[in] Pointer to the structure of the [MqlTradeRequest](#) type, which describes the required trade action.

result

[in,out] Pointer to the structure of the [MqlTradeCheckResult](#) type, to which the check result will be placed.

Return Value

If funds are not enough for the operation, or parameters are filled out incorrectly, the function returns false. In case of a successful basic check of structures (check of pointers), it returns true. **However, this is not an indication that the requested trade operation is sure to be successfully executed.** For a more detailed description of the function execution result, analyze the fields of the *result* structure.

In order to obtain information about the [error](#), call the [GetLastError\(\)](#) function.

See also

[OrderSend\(\)](#), [Trade Operation Types](#), [Trade Request Structure](#), [Structure of Request Check Results](#), [Structure of a Trade Request Result](#)

OrderSend

The OrderSend() function is used for executing [trade operations](#) by sending [requests](#) to a trade server.

```
bool OrderSend(
    MqlTradeRequest& request,      // query structure
    MqlTradeResult& result        // structure of the answer
);
```

Parameters

request

[in] Pointer to a structure of [MqlTradeRequest](#) type describing the trade activity of the client.

result

[in,out] Pointer to a structure of [MqlTradeResult](#) type describing the result of trade operation in case of a successful completion (if true is returned).

Return Value

In case of a successful basic check of structures (index checking) returns true. **However, this is not a sign of successful execution of a trade operation.** For a more detailed description of the function execution result, analyze the fields of *result* structure.

Note

The trade requests go through several stages of checking on a trade server. First of all, it checks if all the required fields of the *request* parameter are filled out correctly. If there are no errors, the server accepts the order for further processing. If the order is successfully accepted by the trade server, the OrderSend() function returns true.

It is recommended to check the request before sending it to a trade server. To check requests, use the [OrderCheck\(\)](#) function. It checks if there are enough funds to execute the trade operation, and returns many useful parameters in the [results of trade request checking](#):

- [return code](#) containing information about errors in the checked request;
- balance value that will appear after the trade operation is executed;
- equity value that will appear after the trade operation is executed;
- floating point value that will appear after the trade operation is executed;
- margin required for the trade operation;
- amount of free equity that will remain after the execution of the trade operation;
- the margin level that will be set after the trade operation is executed;
- comment to the reply code, error description.

It should be noted that the successful completion of the OrderSend() method operation does not always mean a successful deal completion when placing a market order. Check the *retcode* value containing [trade server return code](#), and values of *deal* or *order* field depending on the [type of operation](#) in the returned [result](#).

Each accepted order is stored on the trade server awaiting processing until one of the conditions for its execution occurs:

- expiration,
- appearance of an opposite request,

- order execution when the execution price appears,
- a request to cancel the order is received.

At the moment of the order processing, the trade server sends to the terminal a message about the occurrence of the [Trade](#) event, which can be processed by the [OnTrade\(\)](#) function.

The result of executing the trade request on a server sent by OrderSend() function can be tracked by [OnTradeTransaction](#) handler. It should be noted that OnTradeTransaction handler will be called several times when executing one trade request.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. OnTradeTransaction function will be called for each of these events.

Example:

```
//--- value for ORDER_MAGIC
input long order_magic=55555;

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- make sure that the account is demo
if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
{
    Alert("Script operation is not allowed on a live account!");
    return;
}

//--- place or delete order
if(GetOrdersTotalByMagic(order_magic)==0)
{
    //--- no current orders - place an order
    uint res=SendRandomPendingOrder(order_magic);
    Print("Return code of the trade server ",res);
}
else // there are orders - delete orders
{
    DeleteAllOrdersByMagic(order_magic);
}

//---
}

//+-----+
//| Receives the current number of orders with specified ORDER_MAGIC |
//+-----+
int GetOrdersTotalByMagic(long const magic_number)
{
    ulong order_ticket;
    int total=0;
//--- go through all pending orders
```



```

    for(int i=0;i<OrdersTotal();i++)
        if((order_ticket=OrderGetTicket(i))>0)
            if(magic_number==OrderGetInteger(ORDER_MAGIC)) total++;
//---
    return(total);
}
//+-----+
//| Deletes all pending orders with specified ORDER_MAGIC |
//+-----+
void DeleteAllOrdersByMagic(long const magic_number)
{
    ulong order_ticket;
//--- go through all pending orders
    for(int i=OrdersTotal()-1;i>=0;i--)
        if((order_ticket=OrderGetTicket(i))>0)
            //--- order with appropriate ORDER_MAGIC
            if(magic_number==OrderGetInteger(ORDER_MAGIC))
            {
                MqlTradeResult result={0};
                MqlTradeRequest request={0};
                request.order=order_ticket;
                request.action=TRADE_ACTION_REMOVE;
                OrderSend(request,result);
                //--- write the server reply to log
                Print(__FUNCTION__," ",result.comment," reply code ",result.retcode);
            }
//---
}
//+-----+
//| Sets a pending order in a random way |
//+-----+
uint SendRandomPendingOrder(long const magic_number)
{
//--- prepare a request
    MqlTradeRequest request={0};
    request.action=TRADE_ACTION_PENDING; // setting a pending order
    request.magic=magic_number; // ORDER_MAGIC
    request.symbol=_Symbol; // symbol
    request.volume=0.1; // volume in 0.1 lots
    request.sl=0; // Stop Loss is not specified
    request.tp=0; // Take Profit is not specified
//--- form the order type
    request.type=GetRandomType(); // order type
//--- form the price for the pending order
    request.price=GetRandomPrice(request.type); // open price
//--- send a trade request
    MqlTradeResult result={0};
    OrderSend(request,result);
//--- write the server reply to log

```



```

Print(__FUNCTION__,":",result.comment);
if(result.retcode==10016) Print(result.bid,result.ask,result.price);
//--- return code of the trade server reply
return result.retcode;
}
//+-----+
//| Returns type of a pending order in a random way |
//+-----+
ENUM_ORDER_TYPE GetRandomType()
{
    int t=MathRand()%4;
//--- 0<=t<4
    switch(t)
    {
        case(0):return(ORDER_TYPE_BUY_LIMIT);
        case(1):return(ORDER_TYPE_SELL_LIMIT);
        case(2):return(ORDER_TYPE_BUY_STOP);
        case(3):return(ORDER_TYPE_SELL_STOP);
    }
//--- incorrect value
    return(WRONG_VALUE);
}
//+-----+
//| Returns price in a random way |
//+-----+
double GetRandomPrice(ENUM_ORDER_TYPE type)
{
    int t=(int)type;
//--- stop levels for the symbol
    int distance=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
//--- receive data of the last tick
    MqlTick last_tick={0};
    SymbolInfoTick(_Symbol,last_tick);
//--- calculate price according to the type
    double price;
    if(t==2 || t==5) // ORDER_TYPE_BUY_LIMIT or ORDER_TYPE_SELL_STOP
    {
        price=last_tick.bid; // depart from price Bid
        price=price-(distance+(MathRand()%10)*5)*_Point;
    }
    else // ORDER_TYPE_SELL_LIMIT or ORDER_TYPE_BUY_STOP
    {
        price=last_tick.ask; // depart from price Ask
        price=price+(distance+(MathRand()%10)*5)*_Point;
    }
//---
    return(price);
}

```


See also

[Trade Operation Types](#), [Trade Request Structure](#), [Structure of Request Check Results](#), [Structure of a Trade Request Result](#)

OrderSendAsync

The OrderSendAsync() function is used for conducting asynchronous [trade operations](#) without waiting for the trade server's response to a sent [request](#). The function is designed for high-frequency trading, when under the terms of the trading algorithm it is unacceptable to waste time waiting for a response from the server.

```
bool OrderSendAsync(  
    MqlTradeRequest& request,      // Request structure  
    MqlTradeResult& result        // Response structure  
);
```

Parameters

request

[in] A pointer to a structure of the [MqlTradeRequest](#) type that describes the trade action of the client.

result

[in,out] A pointer to a structure of the [MqlTradeResult](#) type that describes the result of a trade operation in case of successful execution of the function (if true is returned).

Return Value

Returns true if the request is sent to a trade server. In case the request is not sent, it returns false. In case the request is sent, in the *result* variable the response code contains [TRADE_RETCODE_PLACED](#) value (code 10008) - "order placed". Successful execution means only the fact of sending, but does not give any guarantee that the request has reached the trade server and has been accepted for processing. When processing the received request, a trade server sends a reply to a client terminal notifying of change in the current state of positions, orders and deals, which leads to the generation of the [Trade](#) event.

The result of executing the trade request on a server sent by OrderSendAsync() function can be tracked by [OnTradeTransaction](#) handler. It should be noted that OnTradeTransaction handler will be called several times when executing one trade request.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. OnTradeTransaction function will be called for each of these events. To get such a data, the function parameters should be analyzed:

- **trans** - this parameter gets [MqlTradeTransaction](#) structure describing a trade transaction applied to a trade account;
- **request** - this parameter gets [MqlTradeRequest](#) structure describing the trade request resulted in a trade transaction;
- **result** - this parameter gets [MqlTradeResult](#) structure describing a trade request execution result.

Note

In terms of purposes and parameters, the function is similar to [OrderSend\(\)](#), but unlike it, it is asynchronous, i.e. does not hold the program operation while waiting for the function execution result. You can compare the rate of trade operations of these two functions using the sample Expert Advisor.

Example:

```

#property description "Expert Advisor for sending trade requests "
                        " using OrderSendAsync() function.\r\n"
#property description "Handling trading events using"
                        " OnTrade() and OnTradeTransaction() handler functions is displa
#property description "Expert Advisor parameters allow setting Magic Number"
                        " (unique ID) "
#property description "and the mode of displaying messages in Experts log. All details
//--- input parameters
input int   MagicNumber=1234567;      // Expert Advisor ID
input bool  DescriptionModeFull=true; // Detailed output mode
//--- variable for using in HistorySelect() call
datetime history_start;
//+-----+
//| Expert initialization function                                     |
//+-----+
int OnInit()
{
//--- check if autotrading is allowed
    if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
    {
        Alert("Autotrading in the terminal is disabled, Expert Advisor will be removed.");
        ExpertRemove();
        return(-1);
    }
//--- unable to trade on a real account
    if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
    {
        Alert("Expert Advisor cannot trade on a real account!");
        ExpertRemove();
        return(-2);
    }
//--- check if it is possible to trade on this account (for example, trading is impos
    if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
    {
        Alert("Trading on this account is disabled");
        ExpertRemove();
        return(-3);
    }
//--- save the time of launching the Expert Advisor for receiving trading history
    history_start=TimeCurrent();
//---
    CreateBuySellButtons();
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function                                 |
//+-----+

```



```

void OnDeinit(const int reason)
{
    //--- delete all graphical objects
    ObjectDelete(0,"Buy");
    ObjectDelete(0,"Sell");
    //---
}
//+-----+
//| TradeTransaction function |
//+-----+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
{
    //--- heading named after trading event's handler function
    Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
    //--- receive transaction type as enumeration value
    ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
    //--- if transaction is a result of request handling
    if(type==TRADE_TRANSACTION_REQUEST)
    {
        //--- display transaction name
        Print(EnumToString(type));
        //--- then display the string description of the handled request
        Print("-----RequestDescription\r\n",
            RequestDescription(request,DescriptionModeFull));
        //--- and show description of the request result
        Print("----- ResultDescription\r\n",
            TradeResultDescription(result,DescriptionModeFull));
    }
    else // display full description of the transaction for transactions of another type
    {
        Print("----- TransactionDescription\r\n",
            TransactionDescription(trans,DescriptionModeFull));
    }
    //---
}
//+-----+
//| Trade function |
//+-----+
void OnTrade()
{
    //--- static members for storing trading account status
    static int prev_positions=0,prev_orders=0,prev_deals=0,prev_history_orders=0;
    //--- request trading history
    bool update=HistorySelect(history_start,TimeCurrent());
    PrintFormat("HistorySelect(%s , %s) = %s",
        TimeToString(history_start),TimeToString(TimeCurrent()),(string)update);
    //--- heading named after trading event's handler function

```



```

    Print("> ", __FUNCTION__, " at ", TimeToString(TimeCurrent(), TIME_SECONDS));
//--- display handler's name and the number of orders at the moment of handling
    int curr_positions=PositionsTotal();
    int curr_orders=OrdersTotal();
    int curr_deals=HistoryOrdersTotal();
    int curr_history_orders=HistoryDealsTotal();
//--- display the number of orders, positions, deals, as well as changes in parentheses
    PrintFormat("PositionsTotal() = %d (%d)",
                curr_positions, (curr_positions-prev_positions));
    PrintFormat("OrdersTotal() = %d (%d)",
                curr_orders, curr_orders-prev_orders);
    PrintFormat("HistoryOrdersTotal() = %d (%d)",
                curr_deals, curr_deals-prev_deals);
    PrintFormat("HistoryDealsTotal() = %d (%d)",
                curr_history_orders, curr_history_orders-prev_history_orders);
//--- insert a string break to view the log more conveniently
    Print("");
//--- save the account status
    prev_positions=curr_positions;
    prev_orders=curr_orders;
    prev_deals=curr_deals;
    prev_history_orders=curr_history_orders;
//---
}
//+-----+
//| ChartEvent function |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- handling CHARTEVENT_CLICK event ("Clicking the chart")
    if(id==CHARTEVENT_OBJECT_CLICK)
    {
        Print("> ", __FUNCTION__, ": sparam = ", sparam);
        //--- minimum volume for a deal
        double volume_min=SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_MIN);
        //--- if "Buy" button is pressed, then buy
        if(sparam=="Buy")
        {
            PrintFormat("Buy %s %G lot", _Symbol, volume_min);
            BuyAsync(volume_min);
            //--- unpress the button
            ObjectSetInteger(0, "Buy", OBJPROP_STATE, false);
        }
        //--- if "Sell" button is pressed, then sell
        if(sparam=="Sell")
        {

```



```

        PrintFormat("Sell %s %G lot",_Symbol,volume_min);
        SellAsync(volume_min);
        //--- unpress the button
        ObjectSetInteger(0,"Sell",OBJPROP_STATE,false);
    }
    ChartRedraw();
}
//---
}
//+-----+
//| Returns the text description of a transaction |
//+-----+
string TransactionDescription(const MqlTradeTransaction &trans,
                             const bool detailed=true)
{
    //--- prepare a string for returning from the function
    string desc=EnumToString(trans.type)+"\r\n";
    //--- all possible data is added in detailed mode
    if(detailed)
    {
        desc+="Symbol: "+trans.symbol+"\r\n";
        desc+="Deal ticket: "+(string)trans.deal+"\r\n";
        desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
        desc+="Order ticket: "+(string)trans.order+"\r\n";
        desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
        desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
        desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
        desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
        desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
        desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
        desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
        desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
        desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
    }
    //--- return a received string
    return desc;
}
//+-----+
//| Returns the text description of the trade request |
//+-----+
string RequestDescription(const MqlTradeRequest &request,
                          const bool detailed=true)
{
    //--- prepare a string for returning from the function
    string desc=EnumToString(request.action)+"\r\n";
    //--- add all available data in detailed mode
    if(detailed)
    {
        desc+="Symbol: "+request.symbol+"\r\n";
    }
}

```



```

        desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
        desc+="Order ticket: "+(string)request.order+"\r\n";
        desc+="Order type: "+EnumToString(request.type)+"\r\n";
        desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
        desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
        desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
        desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
        desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
        desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
        desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
        desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
        desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
        desc+="Comment: "+request.comment+"\r\n";
    }

    //-- return the received string
    return desc;
}

//+-----+
//| Returns the text description of request handling result |
//+-----+
string TradeResultDescription(const MqlTradeResult &result,
                             const bool detailed=true)
{
    //-- prepare the string for returning from the function
    string desc="Retcode "+(string)result.retcode+"\r\n";
    //-- add all available data in detailed mode
    if(detailed)
    {
        desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
        desc+="Order ticket: "+(string)result.order+"\r\n";
        desc+="Deal ticket: "+(string)result.deal+"\r\n";
        desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
        desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
        desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
        desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
        desc+="Comment: "+result.comment+"\r\n";
    }

    //-- return the received string
    return desc;
}

//+-----+
//| Create two buttons for buying and selling |
//+-----+
void CreateBuySellButtons()
{
    //-- check the object named "Buy"
    if(ObjectFind(0,"Buy")>=0)
    {
        //-- if the found object is not a button, delete it

```



```

        if(ObjectGetInteger(0,"Buy",OBJPROP_TYPE)!=OBJ_BUTTON)
            ObjectDelete(0,"Buy");
    }
    else
        ObjectCreate(0,"Buy",OBJ_BUTTON,0,0,0); // create "Buy" button
//--- configure "Buy" button
    ObjectSetInteger(0,"Buy",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
    ObjectSetInteger(0,"Buy",OBJPROP_XDISTANCE,100);
    ObjectSetInteger(0,"Buy",OBJPROP_YDISTANCE,50);
    ObjectSetInteger(0,"Buy",OBJPROP_XSIZE,70);
    ObjectSetInteger(0,"Buy",OBJPROP_YSIZE,30);
    ObjectSetString(0,"Buy",OBJPROP_TEXT,"Buy");
    ObjectSetInteger(0,"Buy",OBJPROP_COLOR,clrRed);
//--- check presence of the object named "Sell"
    if(ObjectFind(0,"Sell")>=0)
    {
        //--- if the found object is not a button, delete it
        if(ObjectGetInteger(0,"Sell",OBJPROP_TYPE)!=OBJ_BUTTON)
            ObjectDelete(0,"Sell");
    }
    else
        ObjectCreate(0,"Sell",OBJ_BUTTON,0,0,0); // create "Sell" button
//--- configure "Sell" button
    ObjectSetInteger(0,"Sell",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
    ObjectSetInteger(0,"Sell",OBJPROP_XDISTANCE,100);
    ObjectSetInteger(0,"Sell",OBJPROP_YDISTANCE,100);
    ObjectSetInteger(0,"Sell",OBJPROP_XSIZE,70);
    ObjectSetInteger(0,"Sell",OBJPROP_YSIZE,30);
    ObjectSetString(0,"Sell",OBJPROP_TEXT,"Sell");
    ObjectSetInteger(0,"Sell",OBJPROP_COLOR,clrBlue);
//--- perform forced update of the chart to see the buttons immediately
    ChartRedraw();
//---
}
//+-----+
//| Buy using OrderSendAsync() asynchronous function |
//+-----+
void BuyAsync(double volume)
{
//--- prepare the request
    MqlTradeRequest req={0};
    req.action      =TRADE_ACTION_DEAL;
    req.symbol      =_Symbol;
    req.magic       =MagicNumber;
    req.volume      =0.1;
    req.type        =ORDER_TYPE_BUY;
    req.price       =SymbolInfoDouble(req.symbol,SYMBOL_ASK);
    req.deviation    =10;
    req.comment     ="Buy using OrderSendAsync()";

```



```

MqlTradeResult res={0};
if(!OrderSendAsync(req,res))
{
    Print(__FUNCTION__,": error ",GetLastError()," ", retcode = ",res.retcode);
}
//---
}
//+-----+
//| Sell using OrderSendAsync() asynchronous function |
//+-----+
void SellAsync(double volume)
{
//--- prepare the request
MqlTradeRequest req={0};
req.action      =TRADE_ACTION_DEAL;
req.symbol      =_Symbol;
req.magic       =MagicNumber;
req.volume      =0.1;
req.type        =ORDER_TYPE_SELL;
req.price       =SymbolInfoDouble(req.symbol,SYMBOL_BID);
req.deviation    =10;
req.comment     ="Sell using OrderSendAsync()";
MqlTradeResult res={0};
if(!OrderSendAsync(req,res))
{
    Print(__FUNCTION__,": error ",GetLastError()," ", retcode = ",res.retcode);
}
//---
}
//+-----+

```

Example of displaying messages in "Experts" log:

```

12:52:52   ExpertAdvisor (EURUSD,H1)   => OnChartEvent: sparam = Sell
12:52:52   ExpertAdvisor (EURUSD,H1)   Sell EURUSD 0.01 lot
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_REQUEST
12:52:52   ExpertAdvisor (EURUSD,H1)   -----RequestDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_ACTION_DEAL
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Magic Number: 1234567
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order filling: ORDER_FILLING_FOK
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Deviation points: 10
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0

```



```

12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Limit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1) Comment: Sell using OrderSendAsync()
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) ----- ResultDescription
12:52:52 ExpertAdvisor (EURUSD,H1) Retcode 10009
12:52:52 ExpertAdvisor (EURUSD,H1) Request ID: 2
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 15048668
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Ask: 1.29319
12:52:52 ExpertAdvisor (EURUSD,H1) Bid: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Comment:
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+1)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+2)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+2)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_ORDER_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_ORDER_DELETE
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED

```



```

12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_HISTORY_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_FILLED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_DEAL_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 15048668
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC

```



```
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
```


PositionsTotal

Returns the number of open positions.

```
int PositionsTotal();
```

Return Value

Value of [int](#) type.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

See also

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

PositionGetSymbol

Returns the symbol corresponding to the open position and automatically selects the position for further working with it using functions [PositionGetDouble](#), [PositionGetInteger](#), [PositionGetString](#).

```
string PositionGetSymbol(  
    int index    // Number in the list of positions  
);
```

Parameters

index

[in] Number of the position in the list of open positions.

Return Value

Value of the [string](#) type. If the position was not found, an empty string will be returned. To get an [error code](#), call the [GetLastError\(\)](#) function.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

See also

[PositionsTotal\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

PositionSelect

Chooses an open position for further working with it. Returns true if the function is successfully completed. Returns false in case of failure. To obtain information about the error, call [GetLastError\(\)](#).

```
bool PositionSelect(  
    string symbol    // Symbol name  
);
```

Parameters

symbol

[in] Name of the financial security.

Return Value

Value of the bool type.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol. In this case, PositionSelect will select a position with the lowest ticket.

Function PositionSelect() copies data about a position into the program environment, and further calls of [PositionGetDouble\(\)](#), [PositionGetInteger\(\)](#) and [PositionGetString\(\)](#) return the earlier copied data. This means that the position itself may no longer exist (or its volume, direction, etc. has changed), but data of this position still can be obtained. To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

See also

[PositionGetSymbol\(\)](#), [PositionsTotal\(\)](#), [Position Properties](#)

PositionSelectByTicket

Selects an open position to work with based on the ticket number specified in the position. If successful, returns true. Returns false if the function failed. Call [GetLastError\(\)](#) for error details.

```
bool PositionSelectByTicket(  
    ulong    ticket    // Position ticket  
);
```

Parameters

ticket

[in] Position ticket.

Return Value

A value of the bool type.

Note

The PositionSelectByTicket() function copies position data to the program environment. Further calls of [PositionGetDouble\(\)](#), [PositionGetInteger\(\)](#) and [PositionGetString\(\)](#) return the previously copied data. Even if a position does not exist already (or its size, direction etc. has changed), the data may still be received sometimes. To make sure that you receive valid position data, it is recommended to call PositionSelect() before you access the data.

See also

[PositionGetSymbol\(\)](#), [PositionsTotal\(\)](#), [Position Properties](#)

PositionGetDouble

The function returns the requested property of an open position, pre-selected using [PositionGetSymbol](#) or [PositionSelect](#). The position property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double PositionGetDouble(
    ENUM_POSITION_PROPERTY_DOUBLE property_id    // Property identifier
);
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variable passed by reference by the last parameter.

```
bool PositionGetDouble(
    ENUM_POSITION_PROPERTY_DOUBLE property_id,    // Property identifier
    double& double_var    // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the [ENUM_POSITION_PROPERTY_DOUBLE](#) enumeration.

double_var

[out] Variable of the double type, accepting the value of the requested property.

Return Value

Value of the [double](#) type. If the function fails, 0 is returned.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call [PositionSelect\(\)](#) right before referring to them.

See also

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

PositionGetInteger

The function returns the requested property of an open position, pre-selected using [PositionGetSymbol](#) or [PositionSelect](#). The position property should be of datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long PositionGetInteger (
    ENUM_POSITION_PROPERTY_INTEGER property_id    // Property identifier
);
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variables passed by reference by the last parameter.

```
bool PositionGetInteger (
    ENUM_POSITION_PROPERTY_INTEGER property_id,    // Property identifier
    long& long_var                                // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the [ENUM_POSITION_PROPERTY_INTEGER](#) enumeration.

long_var

[out] Variable of the long type accepting the value of the requested property.

Return Value

Value of the [long](#) type. If the function fails, 0 is returned.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call [PositionSelect\(\)](#) right before referring to them.

Example:

```
//+-----+
//| Trade function                                     |
//+-----+
void OnTrade ()
{
    //--- check if a position is present and display the time of its changing
```



```

    if(PositionSelect(_Symbol))
    {
//--- receive position ID for further work
        ulong position_ID=PositionGetInteger(POSITION_IDENTIFIER);
        Print(_Symbol," position #",position_ID);
//--- receive the time of position forming in milliseconds since 01.01.1970
        long create_time_msc=PositionGetInteger(POSITION_TIME_MSC);
        PrintFormat("Position #%d  POSITION_TIME_MSC = %i64 milliseconds => %s",position_ID,
            create_time_msc,TimeToString(create_time_msc/1000));
//--- receive the time of the position's last change in seconds since 01.01.1970
        long update_time_sec=PositionGetInteger(POSITION_TIME_UPDATE);
        PrintFormat("Position #%d  POSITION_TIME_UPDATE = %i64 seconds => %s",
            position_ID,update_time_sec,TimeToString(update_time_sec));
//--- receive the time of the position's last change in milliseconds since 01.01.1970
        long update_time_msc=PositionGetInteger(POSITION_TIME_UPDATE_MSC);
        PrintFormat("Position #%d  POSITION_TIME_UPDATE_MSC = %i64 milliseconds => %s",
            position_ID,update_time_msc,TimeToString(update_time_msc/1000));
    }
//---
}

```

See also

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

PositionGetString

The function returns the requested property of an open position, pre-selected using [PositionGetSymbol](#) or [PositionSelect](#). The position property should be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string PositionGetString(
    ENUM_POSITION_PROPERTY_STRING property_id    // Property identifier
);
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variables passed by reference by the last parameter.

```
bool PositionGetString(
    ENUM_POSITION_PROPERTY_STRING property_id,    // Property identifier
    string& string_var    // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the [ENUM_POSITION_PROPERTY_STRING](#) enumeration.

string_var

[out] Variable of the string type accepting the value of the requested property.

Return Value

Value of the [string](#) type. If the function fails, an empty string is returned.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call [PositionSelect\(\)](#) right before referring to them.

See also

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

PositionGetTicket

The function returns the ticket of a position with the specified index in the list of open positions and automatically selects the position to work with using functions [PositionGetDouble](#), [PositionGetInteger](#), [PositionGetString](#).

```
ulong PositionGetTicket (
    int index    // The number of a position in the list
);
```

Parameters

index

[in] The index of a position in the list of open positions, numeration starts with 0.

Return Value

The ticket of the position. Returns 0 if the function fails.

Note

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call [PositionSelect\(\)](#) right before referring to them.

See also

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Position Properties](#)

OrdersTotal

Returns the number of current orders.

```
int OrdersTotal();
```

Return Value

Value of the [int](#) type.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal. An order is a request to conduct a [transaction](#), while a position is a result of one or more [deals](#).

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

See also

[OrderSelect\(\)](#), [OrderGetTicket\(\)](#), [Order Properties](#)

OrderGetTicket

Returns ticket of a corresponding order and automatically selects the order for further working with it using functions.

```
ulong OrderGetTicket(  
    int index    // Number in the list of orders  
);
```

Parameters

index

[in] Number of an order in the list of current orders.

Return Value

Value of the [ulong](#) type. If the function fails, 0 is returned.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal. An order is a request to conduct a [transaction](#), while a position is a result of one or more [deals](#).

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

Function OrderGetTicket() copies data about an order into the program environment, and further calls of [OrderGetDouble\(\)](#), [OrderGetInteger\(\)](#), [OrderGetString\(\)](#) return the earlier copied data. This means that the order itself may no longer exist (or its open price, Stop Loss/Take Profit levels or expiration has changed), but data of this order still can be obtained. To ensure receipt of fresh data about an order, it is recommended to call OrderGetTicket() right before referring to them.

Example:

```
void OnStart()  
{  
    //--- variables for returning values from order properties  
    ulong    ticket;  
    double   open_price;  
    double   initial_volume;  
    datetime time_setup;  
    string    symbol;  
    string    type;  
    long     order_magic;  
    long     positionID;  
    //--- number of current pending orders  
    uint     total=OrdersTotal();
```



```

//--- go through orders in a loop
for(uint i=0;i<total;i++)
{
    //--- return order ticket by its position in the list
    if((ticket=OrderGetTicket(i))>0)
    {
        //--- return order properties
        open_price    =OrderGetDouble(ORDER_PRICE_OPEN);
        time_setup    =(datetime)OrderGetInteger(ORDER_TIME_SETUP);
        symbol        =OrderGetString(ORDER_SYMBOL);
        order_magic    =OrderGetInteger(ORDER_MAGIC);
        positionID    =OrderGetInteger(ORDER_POSITION_ID);
        initial_volume=OrderGetDouble(ORDER_VOLUME_INITIAL);
        type          =EnumToString(ENUM_ORDER_TYPE(OrderGetInteger(ORDER_TYPE)));
        //--- prepare and show information about the order
        printf("#ticket %d %s %G %s at %G was set up at %s",
            ticket,                // order ticket
            type,                  // type
            initial_volume,        // placed volume
            symbol,                // symbol
            open_price,            // specified open price
            TimeToString(time_setup)// time of order placing
        );
    }
}
//---
}

```

See also

[OrdersTotal\(\)](#), [OrderSelect\(\)](#), [OrderGetInteger\(\)](#)

OrderSelect

Selects an order to work with. Returns true if the function has been successfully completed. Returns false if the function completion has failed. For more information about an error call [GetLastError\(\)](#).

```
bool OrderSelect(  
    ulong ticket    // Order ticket  
);
```

Parameters

ticket

[in] Order ticket.

Return Value

Value of the bool type.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

Function OrderSelect() copies data about an order into the program environment, and further calls of [OrderGetDouble\(\)](#), [OrderGetInteger\(\)](#), [OrderGetString\(\)](#) return the earlier copied data. This means that the order itself may no longer exist (or its open price, Stop Loss/Take Profit levels or expiration has changed), but data of this order still can be obtained. To ensure receipt of fresh data about an order, it is recommended to call OrderSelect() right before referring to them.

See also

[OrderGetInteger\(\)](#), [OrderGetDouble\(\)](#), [OrderGetString\(\)](#), [OrderCalcProfit\(\)](#), [OrderGetTicket\(\)](#), [OrderProperties](#)

OrderGetDouble

Returns the requested property of an order, pre-selected using [OrderGetTicket](#) or [OrderSelect](#). The order property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double OrderGetDouble(
    ENUM_ORDER_PROPERTY_DOUBLE property_id // Property identifier
);
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable passed by reference by the last parameter.

```
bool OrderGetDouble(
    ENUM_ORDER_PROPERTY_DOUBLE property_id, // Property identifier
    double& double_var // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_DOUBLE](#) enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the [double](#) type. If the function fails, 0 is returned.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call [OrderSelect\(\)](#) right before referring to them.

See also

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Order Properties](#)

OrderGetInteger

Returns the requested order property, pre-selected using [OrderGetTicket](#) or [OrderSelect](#). Order property must be of the datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long OrderGetInteger(
    ENUM_ORDER_PROPERTY_INTEGER property_id // Property identifier
);
```

2. Returns true or false depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool OrderGetInteger(
    ENUM_ORDER_PROPERTY_INTEGER property_id, // Property identifier
    long& long_var // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_INTEGER](#) enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the [long](#) type. If the function fails, 0 is returned.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call [OrderSelect\(\)](#) right before referring to them.

See also

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Order Properties](#)

OrderGetString

Returns the requested order property, pre-selected using [OrderGetTicket](#) or [OrderSelect](#). The order property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string OrderGetString(
    ENUM_ORDER_PROPERTY_STRING property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool OrderGetString(
    ENUM_ORDER_PROPERTY_STRING property_id, // Property identifier
    string& string_var // Here we accept the property value
);
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_STRING](#) enumeration.

string_var

[out] Variable of the string type that accepts the value of the requested property.

Return Value

Value of the [string](#) type.

Note

Do not confuse current [pending orders](#) with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call [OrderSelect\(\)](#) right before referring to them.

See also

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Order Properties](#)

HistorySelect

Retrieves the history of deals and orders for the specified period of server time.

```
bool HistorySelect(
    datetime from_date,    // From date
    datetime to_date       // To date
);
```

Parameters

from_date

[in] Start date of the request.

to_date

[in] End date of the request.

Return Value

It returns true if successful, otherwise returns false.

Note

HistorySelect() creates a list of orders and a list of trades in a mql5-program, for further referring to the list elements using corresponding functions. The deals list size can be returned using the [HistoryDealsTotal\(\)](#) function; the size of the list of orders in the history can be obtained using [HistoryOrdersTotal\(\)](#). Selection in the list of orders should be better performed by [HistoryOrderGetTicket\(\)](#), for items in the list of deals [HistoryDealGetTicket\(\)](#) suits better.

After using [HistoryOrderSelect\(\)](#), the list of history orders available to the mql5 program is reset and filled again by the found order, if [the search of an order by the ticket](#) has been completed successfully. The same applies to the list of deals available to the mql5 program - it is reset by [HistoryDealSelect\(\)](#) and filled again in case of a successful receipt of a deal by ticket number.

Example:

```
void OnStart()
{
    color BuyColor =clrBlue;
    color SellColor=clrRed;
    //--- request trade history
    HistorySelect(0,TimeCurrent());
    //--- create objects
    string name;
    uint total=HistoryDealsTotal();
    ulong ticket=0;
    double price;
    double profit;
    datetime time;
    string symbol;
    long type;
    long entry;
    //--- for all deals
```



```

for(uint i=0;i<total;i++)
{
    //--- try to get deals ticket
    if((ticket=HistoryDealGetTicket(i))>0)
    {
        //--- get deals properties
        price =HistoryDealGetDouble(ticket,DEAL_PRICE);
        time  =(datetime)HistoryDealGetInteger(ticket,DEAL_TIME);
        symbol=HistoryDealGetString(ticket,DEAL_SYMBOL);
        type  =HistoryDealGetInteger(ticket,DEAL_TYPE);
        entry =HistoryDealGetInteger(ticket,DEAL_ENTRY);
        profit=HistoryDealGetDouble(ticket,DEAL_PROFIT);
        //--- only for current symbol
        if(price && time && symbol==Symbol())
        {
            //--- create price object
            name="TradeHistory_Deal_"+string(ticket);
            if(entry) ObjectCreate(0,name,OBJ_ARROW_RIGHT_PRICE,0,time,price,0,0);
            else      ObjectCreate(0,name,OBJ_ARROW_LEFT_PRICE,0,time,price,0,0);
            //--- set object properties
            ObjectSetInteger(0,name,OBJPROP_SELECTABLE,0);
            ObjectSetInteger(0,name,OBJPROP_BACK,0);
            ObjectSetInteger(0,name,OBJPROP_COLOR,type?BuyColor:SellColor);
            if(profit!=0) ObjectSetString(0,name,OBJPROP_TEXT,"Profit: "+string(profit));
        }
    }
}

//--- apply on chart
ChartRedraw();
}

```

See also

[HistoryOrderSelect\(\)](#), [HistoryDealSelect\(\)](#)

HistorySelectByPosition

Retrieves the history of deals and orders having the specified position identifier.

```
bool HistorySelectByPosition(  
    long position_id // position identifier - POSITION\_IDENTIFIER  
);
```

Parameters

position_id

[in] Position identifier that is set to every executed order and every deal.

Return Value

It returns true if successful, otherwise returns false.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

HistorySelectByPosition() creates in a mql5 program a list of orders and a list of deals with a specified [position identifier](#) for further reference to the elements of the list using the appropriate functions. To know the size of the list of deals, use function [HistoryDealsTotal\(\)](#), the size of the list of orders in the history can be obtained using [HistoryOrdersTotal\(\)](#). To run through elements of the orders list, use [HistoryOrderGetTicket\(\)](#), for elements of the deals list - [HistoryDealGetTicket\(\)](#).

After using [HistoryOrderSelect\(\)](#), list of history orders available to the mql5 program is reset and filled again with the found order, if [search of an order by its ticket](#) was successful. The same refers to the list of deals available to the mql5 program - it is reset by function [HistoryDealSelect\(\)](#) and is filled out again if a deal was found successfully by the ticket number.

See also

[HistorySelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Order Properties](#)

HistoryOrderSelect

Selects an order from the history for further calling it through appropriate functions. It returns true if the function has been successfully completed. Returns false if the function has failed. For more details on error call [GetLastError\(\)](#).

```
bool HistoryOrderSelect (
    ulong ticket    // Order ticket
);
```

Parameters

ticket

[in] Order ticket.

Return Value

Returns true if successful, otherwise false.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

HistoryOrderSelect() clears in a mql5-program the list of orders from a history, available for calls, and copies to it a single order, if the execution of HistoryOrderSelect () has been completed successfully. If you need to go through all deals selected by [HistorySelect\(\)](#), you should better use [HistoryOrderGetTicket\(\)](#).

See also

[HistorySelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Order Properties](#)

HistoryOrdersTotal

Returns the number of orders in the history. Prior to calling HistoryOrdersTotal(), first it is necessary to receive the history of deals and orders using the [HistorySelect\(\)](#) or [HistorySelectByPosition\(\)](#) function.

```
int HistoryOrdersTotal();
```

Return Value

Value of the [int](#) type.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

See also

[HistorySelect\(\)](#), [HistoryOrderSelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Order Properties](#)

HistoryOrderGetTicket

Return the ticket of a corresponding order in the history. Prior to calling `HistoryOrderGetTicket()`, first it is necessary to receive the history of deals and orders using the [HistorySelect\(\)](#) or [HistorySelectByPosition\(\)](#) function.

```
ulong HistoryOrderGetTicket(  
    int index      // Number in the list of orders  
);
```

Parameters

index

[in] Number of the order in the list of orders.

Return Value

Value of the [ulong](#) type. If the function fails, 0 is returned.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
void OnStart()  
{  
    datetime from=0;  
    datetime to=TimeCurrent();  
    //--- request the entire history  
    HistorySelect(from,to);  
    //--- variables for returning values from order properties  
    ulong    ticket;  
    double   open_price;  
    double   initial_volume;  
    datetime time_setup;  
    datetime time_done;  
    string    symbol;  
    string    type;  
    long      order_magic;  
    long      positionID;  
    //--- number of current pending orders  
    uint      total=HistoryOrdersTotal();  
    //--- go through orders in a loop  
    for(uint i=0;i<total;i++)  
    {  
        //--- return order ticket by its position in the list  
        if((ticket=HistoryOrderGetTicket(i))>0)  
        {  
            //--- return order properties
```



```

open_price      =HistoryOrderGetDouble(ticket,ORDER_PRICE_OPEN);
time_setup      =(datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_SETUP);
time_done       =(datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_DONE);
symbol          =HistoryOrderGetString(ticket,ORDER_SYMBOL);
order_magic     =HistoryOrderGetInteger(ticket,ORDER_MAGIC);
positionID      =HistoryOrderGetInteger(ticket,ORDER_POSITION_ID);
initial_volume  =HistoryOrderGetDouble(ticket,ORDER_VOLUME_INITIAL);
type            =GetOrderType(HistoryOrderGetInteger(ticket,ORDER_TYPE));
//--- prepare and show information about the order
printf("#ticket %d %s %G %s at %G was set up at %s => done at %s, pos ID=%d",
        ticket,                // order ticket
        type,                  // type
        initial_volume,        // placed volume
        symbol,                // symbol
        open_price,            // specified open price
        TimeToString(time_setup), // time of order placing
        TimeToString(time_done), // time of order execution or deletion
        positionID             // ID of a position , to which the deal of t
    );
}
}
//---
}
//+-----+
//| Returns the string name of the order type |
//+-----+
string GetOrderType(long type)
{
    string str_type="unknown operation";
    switch(type)
    {
        case (ORDER_TYPE_BUY):          return("buy");
        case (ORDER_TYPE_SELL):          return("sell");
        case (ORDER_TYPE_BUY_LIMIT):     return("buy limit");
        case (ORDER_TYPE_SELL_LIMIT):    return("sell limit");
        case (ORDER_TYPE_BUY_STOP):      return("buy stop");
        case (ORDER_TYPE_SELL_STOP):     return("sell stop");
        case (ORDER_TYPE_BUY_STOP_LIMIT): return("buy stop limit");
        case (ORDER_TYPE_SELL_STOP_LIMIT):return("sell stop limit");
    }
    return(str_type);
}

```

See also

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Order Properties](#)

HistoryOrderGetDouble

Returns the requested order property. The order property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double HistoryOrderGetDouble(  
    ulong ticket_number, // Ticket  
    ENUM_ORDER_PROPERTY_DOUBLE property_id // Property identifier  
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryOrderGetDouble(  
    ulong ticket_number, // Ticket  
    ENUM_ORDER_PROPERTY_DOUBLE property_id, // Property identifier  
    double& double_var // Here we accept the property value  
);
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_DOUBLE](#) enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the [double](#) type.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

See also

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Order Properties](#)

HistoryOrderGetInteger

Returns the requested property of an order. The order property must be of datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long HistoryOrderGetInteger(
    ulong          ticket_number,    // Ticket
    ENUM_ORDER_PROPERTY_INTEGER property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryOrderGetInteger(
    ulong          ticket_number,    // Ticket
    ENUM_ORDER_PROPERTY_INTEGER property_id, // Property identifier
    long&          long_var         // Here we accept the property value
);
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_INTEGER](#) enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the [long](#) type.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
//+-----+
//| Trade function |
//+-----+
void OnTrade()
{
    //--- receive the last order's ticket from week's trading history
    ulong last_order=GetLastOrderTicket();
    if(HistoryOrderSelect(last_order))
```



```

{
    //--- time of placing an order in milliseconds since 01.01.1970
    long time_setup_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_SETUP_MSC);
    PrintFormat("Order #d ORDER_TIME_SETUP_MSC=%i64 => %s",
                last_order,time_setup_msc,TimeToString(time_setup_msc/1000));
    //--- order execution/cancellation time in milliseconds since 01.01.1970
    long time_done_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_DONE_MSC);
    PrintFormat("Order #d ORDER_TIME_DONE_MSC=%i64 => %s",
                last_order,time_done_msc,TimeToString(time_done_msc/1000));
}
else // notify on failure
    PrintFormat("HistoryOrderSelect() failed for #d. Error code=%d",
                last_order,GetLastError());

//---
}
//+-----+
//| Returns the last order ticket in history or -1 |
//+-----+
ulong GetLastOrderTicket()
{
    //--- request history for the last 7 days
    if(!GetTradeHistory(7))
    {
        //--- notify on unsuccessful call and return -1
        Print(__FUNCTION__," HistorySelect() returned false");
        return -1;
    }
    //---
    ulong first_order,last_order,orders=HistoryOrdersTotal();
    //--- work with orders if there are any
    if(orders>0)
    {
        Print("Orders = ",orders);
        first_order=HistoryOrderGetTicket(0);
        PrintFormat("first_order = %d",first_order);
        if(orders>1)
        {
            last_order=HistoryOrderGetTicket((int)orders-1);
            PrintFormat("last_order = %d",last_order);
            return last_order;
        }
        return first_order;
    }
    //--- no order found, return -1
    return -1;
}
//+-----+
//| Requests history for the last days and returns false in case of failure |

```



```
//+-----+
bool GetTradeHistory(int days)
{
    //--- set a week period to request trade history
    datetime to=TimeCurrent();
    datetime from=to-days*PeriodSeconds(PERIOD_D1);
    ResetLastError();
    //--- make a request and check the result
    if(!HistorySelect(from,to))
    {
        Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
        return false;
    }
    //--- history received successfully
    return true;
}
```

See also

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Order Properties](#)

HistoryOrderGetString

Returns the requested property of an order. The order property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string HistoryOrderGetString(
    ulong          ticket_number,    // Ticket
    ENUM_ORDER_PROPERTY_STRING property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryOrderGetString(
    ulong          ticket_number,    // Ticket
    ENUM_ORDER_PROPERTY_STRING property_id, // Property identifier
    string&        string_var       // Here we accept the property value
);
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the [ENUM_ORDER_PROPERTY_STRING](#) enumeration.

string_var

[out] Variable of the string type.

Return Value

Value of the [string](#) type.

Note

Do not confuse orders of a trading history with current [pending orders](#) that appear on the "Trade" tab of the "Toolbox" bar. The list of [orders](#) that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

See also

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Order Properties](#)

HistoryDealSelect

Selects a deal in the history for further calling it through appropriate functions. It returns true if the function has been successfully completed. Returns false if the function has failed. For more details on error call [GetLastError\(\)](#).

```
bool HistoryDealSelect(  
    ulong ticket    // Deal ticket  
);
```

Parameters

ticket

[in] Deal ticket.

Return Value

Returns true if successful, otherwise false.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

HistoryDealSelect() clears in a mql5-program the list of deals available for reference, and copies the single deal, if the execution of HistoryDealSelect() has been completed successfully. If you need to go through all deals selected by the [HistorySelect\(\)](#) function, you should better use [HistoryDealGetTicket\(\)](#).

See also

[HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Deal Properties](#)

HistoryDealsTotal

Returns the number of deal in history. Prior to calling HistoryDealsTotal(), first it is necessary to receive the history of deals and orders using the [HistorySelect\(\)](#) or [HistorySelectByPosition\(\)](#) function.

```
int HistoryDealsTotal();
```

Return Value

Value of the [int](#) type.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

See also

[HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Deal Properties](#)

HistoryDealGetTicket

The function selects a deal for further processing and returns the deal ticket in history. Prior to calling `HistoryDealGetTicket()`, first it is necessary to receive the history of deals and orders using the [HistorySelect\(\)](#) or [HistorySelectByPosition\(\)](#) function.

```
ulong HistoryDealGetTicket (
    int index      // ticket deal
);
```

Parameters

index

[in] Number of a deal in the list of deals

Return Value

Value of the [ulong](#) type. If the function fails, 0 is returned.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
void OnStart()
{
    ulong deal_ticket;           // deal ticket
    ulong order_ticket;         // ticket of the order the deal was executed on
    datetime transaction_time;  // time of a deal execution
    long deal_type ;            // type of a trade operation
    long position_ID;           // position ID
    string deal_description;     // operation description
    double volume;              // operation volume
    string symbol;              // symbol of the deal

    //--- set the start and end date to request the history of deals
    datetime from_date=0;       // from the very beginning
    datetime to_date=TimeCurrent(); // till the current moment
    //--- request the history of deals in the specified period
    HistorySelect(from_date,to_date);
    //--- total number in the list of deals
    int deals=HistoryDealsTotal();
    //--- now process each trade
    for(int i=0;i<deals;i++)
    {
        deal_ticket=             HistoryDealGetTicket(i);
        volume=                  HistoryDealGetDouble(deal_ticket,DEAL_VOLUME);
        transaction_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
        order_ticket=            HistoryDealGetInteger(deal_ticket,DEAL_ORDER);
        deal_type=                HistoryDealGetInteger(deal_ticket,DEAL_TYPE);
        symbol=                   HistoryDealGetString(deal_ticket,DEAL_SYMBOL);
    }
}
```



```

        position_ID=                HistoryDealGetInteger(deal_ticket,DEAL_POSITION_ID);
        deal_description=            GetDealDescription(deal_type,volume,symbol,order_ticket);
        //--- perform fine formatting for the deal number
        string print_index=StringFormat("% 3d",i);
        //--- show information on the deal
        Print(print_index+": deal #",deal_ticket," at ",transaction_time,deal_description);
    }
}

//+-----+
//| Returns the string description of the operation |
//+-----+
string GetDealDescription(long deal_type,double volume,string symbol,long ticket,long position_ID)
{
    string descr;
    //---
    switch(deal_type)
    {
        case DEAL_TYPE_BALANCE:                return ("balance");
        case DEAL_TYPE_CREDIT:                 return ("credit");
        case DEAL_TYPE_CHARGE:                 return ("charge");
        case DEAL_TYPE_CORRECTION:             return ("correction");
        case DEAL_TYPE_BUY:                    descr="buy"; break;
        case DEAL_TYPE_SELL:                   descr="sell"; break;
        case DEAL_TYPE_BONUS:                  return ("bonus");
        case DEAL_TYPE_COMMISSION:              return ("additional commission");
        case DEAL_TYPE_COMMISSION_DAILY:        return ("daily commission");
        case DEAL_TYPE_COMMISSION_MONTHLY:      return ("monthly commission");
        case DEAL_TYPE_COMMISSION_AGENT_DAILY:  return ("daily agent commission");
        case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return ("monthly agent commission");
        case DEAL_TYPE_INTEREST:                return ("interest rate");
        case DEAL_TYPE_BUY_CANCELED:            descr="cancelled buy deal"; break;
        case DEAL_TYPE_SELL_CANCELED:          descr="cancelled sell deal"; break;
    }
    descr=StringFormat("%s %G %s (order #d, position ID %d)",
        descr, // current description
        volume, // deal volume
        symbol, // deal symbol
        ticket, // ticket of the order that caused the deal
        pos_ID // ID of a position, in which the deal is included
    );

    return(descr);
}
//---
}

```

See also

[HistorySelect\(\)](#), [HistoryDealsTotal\(\)](#), [HistoryDealSelect\(\)](#), [Deal Properties](#)

HistoryDealGetDouble

Returns the requested property of a deal. The deal property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double HistoryDealGetDouble(
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_DOUBLE property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryDealGetDouble(
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_DOUBLE property_id, // Property identifier
    double&        double_var       // Here we accept the property value
);
```

Parameters

ticket_number

[in] Deal ticket.

property_id

[in] Identifier of a deal property. The value can be one of the values of the [ENUM_DEAL_PROPERTY_DOUBLE](#) enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the [double](#) type.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

See also

[HistorySelect\(\)](#), [HistoryDealsTotal\(\)](#), [HistoryDealGetTicket\(\)](#), [Deal Properties](#)

HistoryDealGetInteger

Returns the requested property of a deal. The deal property must be of the datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long HistoryDealGetInteger (
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_INTEGER property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryDealGetInteger (
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_INTEGER property_id, // Property identifier
    long&          long_var         // Here we accept the property value
);
```

Parameters

ticket_number

[in] Trade ticket.

property_id

[in] Identifier of the deal property. The value can be one of the values of the [ENUM_DEAL_PROPERTY_INTEGER](#) enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the [long](#) type.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
//+-----+
//| Trade function |
//+-----+
void OnTrade()
{
    //--- receive the last deal's ticket from week's trading history
    ulong last_deal=GetLastDealTicket();
    if(HistoryDealSelect(last_deal))
    {
```



```

    //--- time of deal execution in milliseconds since 01.01.1970
    long deal_time_msc=HistoryDealGetInteger(last_deal,DEAL_TIME_MSC);
    PrintFormat("Deal #d DEAL_TIME_MSC=%i64 => %s",
                last_deal,deal_time_msc,TimeToString(deal_time_msc/1000));
}
else
    PrintFormat("HistoryDealSelect() failed for #d. Error code=%d",
                last_deal,GetLastError());
//---
}
//+-----+
//| Returns the last deal ticket in history or -1 |
//+-----+
ulong GetLastDealTicket()
{
    //--- request history for the last 7 days
    if(!GetTradeHistory(7))
    {
        //--- notify on unsuccessful call and return -1
        Print(__FUNCTION__," HistorySelect() returned false");
        return -1;
    }
    //---
    ulong first_deal,last_deal,deals=HistoryOrdersTotal();
    //--- work with orders if there are any
    if(deals>0)
    {
        Print("Deals = ",deals);
        first_deal=HistoryDealGetTicket(0);
        PrintFormat("first_deal = %d",first_deal);
        if(deals>1)
        {
            last_deal=HistoryDealGetTicket((int)deals-1);
            PrintFormat("last_deal = %d",last_deal);
            return last_deal;
        }
        return first_deal;
    }
    //--- no deal found, return -1
    return -1;
}
//+-----+
//| Requests history for the last days and returns false in case of failure |
//+-----+
bool GetTradeHistory(int days)
{
    //--- set a week period to request trade history
    datetime to=TimeCurrent();
    datetime from=to-days*PeriodSeconds(PERIOD_D1);

```



```
ResetLastError();  
//--- make a request and check the result  
if(!HistorySelect(from,to))  
{  
    Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());  
    return false;  
}  
//--- history received successfully  
return true;  
}
```

See also

[HistoryDealsTotal\(\)](#), [HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Deal Properties](#)

HistoryDealGetString

Returns the requested property of a deal. The deal property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string HistoryDealGetString(
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_STRING property_id // Property identifier
);
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool HistoryDealGetString(
    ulong          ticket_number,    // Ticket
    ENUM_DEAL_PROPERTY_STRING property_id, // Property identifier
    string&        string_var       // Here we accept the property value
);
```

Parameters

ticket_number

[in] Deal ticket.

property_id

[in] Identifier of the deal property. The value can be one of the values of the [ENUM_DEAL_PROPERTY_STRING](#) enumeration.

string_var

[out] Variable of the string type that accepts the value of the requested property.

Return Value

Value of the [string](#) type.

Note

Do not confuse [orders](#), [deals](#) and [positions](#). Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

See also

[HistoryDealsTotal\(\)](#), [HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Deal Properties](#)

Trade Signals

This is the group of functions intended for managing trade signals. The functions allow:

- get information about trade signals, available for copying,
- get and set the signal copy settings,
- subscribe and unsubscribe to the signal copying using MQL5 language functions.

Function	Action
<u>SignalBaseGetDouble</u>	Returns the value of double type property for selected signal
<u>SignalBaseGetInteger</u>	Returns the value of integer type property for selected signal
<u>SignalBaseGetString</u>	Returns the value of string type property for selected signal
<u>SignalBaseSelect</u>	Selects a signal from signals, available in terminal for further working with it
<u>SignalBaseTotal</u>	Returns the total amount of signals, available in terminal
<u>SignalInfoGetDouble</u>	Returns the value of double type property of signal copy settings
<u>SignalInfoGetInteger</u>	Returns the value of integer type property of signal copy settings
<u>SignalInfoGetString</u>	Returns the value of string type property of signal copy settings
<u>SignalInfoSetDouble</u>	Sets the value of double type property of signal copy settings
<u>SignalInfoSetInteger</u>	Sets the value of integer type property of signal copy settings
<u>SignalSubscribe</u>	Subscribes to the trading signal
<u>SignalUnsubscribe</u>	Cancels subscription

SignalBaseGetDouble

Returns the value of [double](#) type property for selected signal.

```
double SignalBaseGetDouble(  
    ENUM_SIGNAL_BASE_DOUBLE    property_id,    // property identifier  
);
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the [ENUM_SIGNAL_BASE_DOUBLE](#) enumeration.

Return Value

The value of [double](#) type property of the selected signal.

SignalBaseGetInteger

Returns the value of [integer](#) type property for selected signal.

```
long SignalBaseGetInteger (
    ENUM_SIGNAL_BASE_INTEGER    property_id,    // property identifier
);
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the [ENUM_SIGNAL_BASE_INTEGER](#) enumeration.

Return Value

The value of [integer](#) type property of the selected signal.

SignalBaseGetString

Returns the value of [string](#) type property for selected signal.

```
string SignalBaseGetString(  
    ENUM_SIGNAL_BASE_STRING    property_id,    // property identifier  
);
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the [ENUM_SIGNAL_BASE_STRING](#) enumeration.

Return Value

The value of [string](#) type property of the selected signal.

SignalBaseSelect

Selects a signal from signals, available in terminal for further working with it.

```
bool SignalBaseSelect(
    int    index    // signal index
);
```

Parameters

index

[in] Signal index in base of trading signals.

Return Value

Returns true if successful, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Example:

```
void OnStart()
{
    //--- get total amount of signals in the terminal
    int total=SignalBaseTotal();
    //--- process all signals
    for(int i=0;i<total;i++)
    {
        //--- select the signal by index
        if(SignalBaseSelect(i))
        {
            //--- get signal properties
            long id    =SignalBaseGetInteger(SIGNAL_BASE_ID);           // signal id
            long pips   =SignalBaseGetInteger(SIGNAL_BASE_PIPS);        // profit in pips
            long subscr =SignalBaseGetInteger(SIGNAL_BASE_SUBSCRIBERS); // number of subscribers
            string name  =SignalBaseGetString(SIGNAL_BASE_NAME);        // signal name
            double price =SignalBaseGetDouble(SIGNAL_BASE_PRICE);       // signal price
            string curr  =SignalBaseGetString(SIGNAL_BASE_CURRENCY);    // signal currency
            //--- print all profitable free signals with subscribers
            if(price==0.0 && pips>0 && subscr>0)
                PrintFormat("id=%d, name=\"%s\", currency=%s, pips=%d, subscribers=%d",id,
                    name,curr,pips,subscr);
        }
        else PrintFormat("Error in call of SignalBaseSelect. Error code=%d",GetLastError());
    }
}
```


SignalBaseTotal

Returns the total amount of signals, available in terminal.

```
int SignalBaseTotal();
```

Return Value

The total amount of signals, available in terminal.

SignalInfoGetDouble

Returns the value of [double](#) type property of signal copy settings.

```
double SignalInfoGetDouble(  
    ENUM_SIGNAL_INFO_DOUBLE    property_id,    // property identifier  
);
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the [ENUM_SIGNAL_INFO_DOUBLE](#) enumeration.

Return Value

The value of [double](#) type property of signal copy settings.

SignalInfoGetInteger

Returns the value of [integer](#) type property of signal copy settings.

```
long  SignalInfoGetInteger (
    ENUM_SIGNAL_INFO_INTEGER  property_id,    // property identifier
);
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the [ENUM_SIGNAL_INFO_INTEGER](#) enumeration.

Return Value

The value of [integer](#) type property of signal copy settings.

SignalInfoGetString

Returns the value of [string](#) type property of signal copy settings.

```
string SignalInfoGetString(  
    ENUM_SIGNAL_INFO_STRING    property_id,    // property identifier  
);
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the [ENUM_SIGNAL_INFO_STRING](#) enumeration.

Return Value

The value of [string](#) type property of signal copy settings.

SignalInfoSetDouble

Sets the value of [double](#) type property of signal copy settings.

```
bool SignalInfoSetDouble(  
    ENUM_SIGNAL_INFO_DOUBLE    property_id,    // property identifier  
    double                     value           // new value  
);
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the [ENUM_SIGNAL_INFO_DOUBLE](#) enumeration.

value

[in] The value of signal copy settings property.

Return Value

Returns true if property has been changed, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

SignalInfoSetInteger

Sets the value of [integer](#) type property of signal copy settings.

```
bool SignalInfoSetInteger (
    ENUM_SIGNAL_INFO_INTEGER    property_id,    // property identifier
    long                        value           // new value
);
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the [ENUM_SIGNAL_INFO_INTEGER](#) enumeration.

value

[in] The value of signal copy settings property.

Return Value

Returns true if property has been changed, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

SignalSubscribe

Subscribes to the trading signal.

```
bool SignalSubscribe(  
    long    signal_id    // signal id  
);
```

Parameters

signal_id

[in] Signal identifier.

Return Value

Returns true if subscription was successful, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

SignalUnsubscribe

Cancels subscription.

```
bool SignalUnsubscribe();
```

Return Value

Returns true if subscription has been canceled successfully, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Global Variables of the Client Terminal

There is a group set of functions for working with global variables.

Global variables of the client terminal should not be mixed up with variables declared in the [global scope](#) of the mql5 program.

Global variables are kept in the client terminal for 4 weeks since the last access, then they will be deleted automatically. An access to a global variable is not only setting of a new value, but reading of the global variable value, as well.

Global variables of the client terminal are accessible simultaneously from all mql5 programs launched in the client terminal.

Function	Action
GlobalVariableCheck	Checks the existence of a global variable with the specified name
GlobalVariableTime	Returns time of the last accessing the global variable
GlobalVariableDel	Deletes a global variable
GlobalVariableGet	Returns the value of a global variable
GlobalVariableName	Returns the name of a global variable by its ordinal number in the list of global variables
GlobalVariableSet	Sets the new value to a global variable
GlobalVariablesFlush	Forcibly saves contents of all global variables to a disk
GlobalVariableTemp	Sets the new value to a global variable, that exists only in the current session of the terminal
GlobalVariableSetOnCondition	Sets the new value of the existing global variable by condition
GlobalVariablesDeleteAll	Deletes global variables with the specified prefix in their names
GlobalVariablesTotal	Returns the total number of global variables

GlobalVariableCheck

Checks the existence of a global variable with the specified name

```
bool GlobalVariableCheck(  
    string name    // Global variable name  
);
```

Parameters

name

[in] Global variable name.

Return Value

Returns true, if the global variable exists, otherwise returns false.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

See also

[GlobalVariableTime\(\)](#)

GlobalVariableTime

Returns the time when the global variable was last accessed.

```
datetime GlobalVariableTime(  
    string name    // name  
);
```

Parameters

name

[in] Name of the global variable.

Return Value

The function returns time of last accessing the specified global variable. Addressing a variable for its value, for example using the [GlobalVariableGet\(\)](#) and [GlobalVariableCheck\(\)](#) functions, also modifies the time of last access. In order to obtain [error](#) details, call the [GetLastError\(\)](#) function.

Note

Global variables exist in the client terminal during 4 weeks since they were called last. After that they are automatically deleted.

See also

[GlobalVariableCheck\(\)](#)

GlobalVariableDel

Deletes a global variable from the client terminal.

```
bool GlobalVariableDel(  
    string name    // Global variable name  
);
```

Parameters

name

[in] Global variable name.

Return Value

If successful, the function returns true, otherwise it returns false. To obtain an information about the [error](#) it is necessary to call the function [GetLastError\(\)](#).

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableGet

Returns the value of an existing global variable of the client terminal. There are 2 variants of the function.

1. Immediately returns the value of the global variable.

```
double GlobalVariableGet(  
    string name      // Global variable name  
);
```

2. Returns true or false depending on the success of the function run. If successful, the global variable of the client terminal is placed in a variable passed by reference in the second parameter.

```
bool GlobalVariableGet(  
    string name,          // Global variable name  
    double& double_var    // This variable will contain the value of the global variable  
);
```

Parameters

name

[in] Global variable name.

double_var

[out] Target variable of the double type, which accepts the value stored in a the global variable of the client terminal.

Return Value

The value of the existing global variable or 0 in case of an [error](#). For more details about the error, call [GetLastError\(\)](#).

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableName

Returns the name of a global variable by its ordinal number.

```
string GlobalVariableName(  
    int index    // Global variable number in the list of global variables  
);
```

Parameters

index

[in] Sequence number in the list of global variables. It should be greater than or equal to 0 and less than [GlobalVariablesTotal\(\)](#).

Return Value

Global variable name by its ordinal number in the list of global variables. For more details about the [error](#), call [GetLastError\(\)](#).

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableSet

Sets a new value for a global variable. If the variable does not exist, the system creates a new global variable.

```
datetime GlobalVariableSet(  
    string  name,          // Global variable name  
    double  value          // Value to set  
);
```

Parameters

name

[in] Global variable name.

value

[in] The new numerical value.

Return Value

If successful, the function returns the last modification time, otherwise 0. For more details about the [error](#), call [GetLastError\(\)](#).

Note

A global variable name should not exceed 63 characters. Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesFlush

Forcibly saves contents of all global variables to a disk.

```
void GlobalVariablesFlush();
```

Return Value

No return value.

Note

The terminal writes all the global variables when the work is over, but data can be lost at a sudden computer operation failure. This function allows independently controlling the process of saving global variables in case of contingency.

GlobalVariableTemp

The function attempts to create a temporary global variable. If the variable doesn't exist, the system creates a new temporary global variable.

```
bool GlobalVariableTemp(  
    string name    // Global variable name  
);
```

Parameters

name

[in] The name of a temporary global variable.

Return Value

If successful, the function returns true, otherwise - false. To get details about the [error](#), you should call the [GetLastError\(\)](#) function.

Note

Temporary global variables exist only while the client terminal is running; after the terminal shutdown they are automatically deleted. Note that during the execution of [GlobalVariablesFlush\(\)](#) temporary global variables are not written to a disk.

After a temporary global variable has been created, it can be accessed and modified the same as [global variable of the client terminal](#).

GlobalVariableSetOnCondition

Sets the new value of the existing global variable if the current value equals to the third parameter `check_value`. If there is no global variable, the function will generate an error `ERR_GLOBALVARIABLE_NOT_FOUND` (4501) and return false.

```
bool GlobalVariableSetOnCondition(  
    string  name,           // Global variable name  
    double  value,          // New value for variable if condition is true  
    double  check_value     // Check value condition  
);
```

Parameters

name

[in] The name of a global variable.

value

[in] New value.

check_value

[in] The value to check the current value of the global variable.

Return Value

If successful, the function returns true, otherwise it returns false. For details about the [error](#) call [GetLastError\(\)](#). If the current value of the global variable is different from `check_value`, the function returns false.

Note

Function provides atomic access to the global variable, so it can be used for providing of a mutex at interaction of several Expert Advisors working simultaneously within one client terminal.

GlobalVariablesDeleteAll

Deletes global variables of the client terminal.

```
int GlobalVariablesDeleteAll(  
    string    prefix_name=NULL,    // All global variables with names beginning with  
    datetime  limit_data=0        // All global variables that were changed before t  
);
```

Parameters

prefix_name=NULL

[in] Name prefix global variables to remove. If you specify a prefix NULL or empty string, then all variables that meet the data criterion will be deleted.

limit_data=0

[in] Date to select global variables by the time of their last modification. The function removes global variables, which were changed before this date. If the parameter is zero, then all variables that meet the first criterion (prefix) are deleted.

Return Value

The number of deleted variables.

Note

If both options are equal to zero (*prefix_name* = NULL and *limit_data* = 0), then function deletes all global variables of the terminal. If both parameters are specified, then it deletes global variables corresponding to both parameters.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesTotal

Returns the total number of global variables of the client terminal.

```
int GlobalVariablesTotal();
```

Return Value

Number of global variables.

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted. Call of a global variable is not only setting a new value, but also reading the value of the global variable.

File Functions

This is a group of functions for working with files.

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means cannot be outside the file sandbox.

There are two directories (with subdirectories) in which working files can be located:

- terminal_data_folder\MQL5\FILES\ (in the terminal menu select to view "File" - "Open the data directory");
- the common folder for all the terminals installed on a computer - usually located in the directory C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\Files.

There is a program method to obtain names of these catalogs using the [TerminalInfoString\(\)](#) function, using the [ENUM_TERMINAL_INFO_STRING](#) enumeration:

```
//--- Folder that stores the terminal data
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
//--- Common folder for all client terminals
string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

Work with files from other directories is prohibited.

File functions allow working with so-called "named pipes". To do this, simply call [FileOpen\(\)](#) function with appropriate parameters.

Function	Action
FileFindFirst	Starts the search of files in a directory in accordance with the specified filter
FileFindNext	Continues the search started by the FileFindFirst() function
FileFindClose	Closes search handle
FileOpen	Opens a file with a specified name and flag
FileDelete	Deletes a specified file
FileFlush	Writes to a disk all data remaining in the input/output file buffer
FileGetInteger	Gets an integer property of a file
FilesEnding	Defines the end of a file in the process of reading
FilesLineEnding	Defines the end of a line in a text file in the process of reading
FileClose	Closes a previously opened file
FilesExist	Checks the existence of a file

<u>FileCopy</u>	Copies the original file from a local or shared folder to another file
<u>FileMove</u>	Moves or renames a file
<u>FileReadArray</u>	Reads arrays of any type except for string from the file of the BIN type
<u>FileReadBool</u>	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type
<u>FileReadDatetime</u>	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value
<u>FileReadDouble</u>	Reads a double value from the current position of the file pointer
<u>FileReadFloat</u>	Reads a float value from the current position of the file pointer
<u>FileReadInteger</u>	Reads int, short or char value from the current position of the file pointer
<u>FileReadLong</u>	Reads a long type value from the current position of the file pointer
<u>FileReadNumber</u>	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string into double value
<u>FileReadString</u>	Reads a string from the current position of a file pointer from a file
<u>FileReadStruct</u>	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer
<u>FileSeek</u>	Moves the position of the file pointer by a specified number of bytes relative to the specified position
<u>FileSize</u>	Returns the size of a corresponding open file
<u>FileTell</u>	Returns the current position of the file pointer of a corresponding open file
<u>FileWrite</u>	Writes data to a file of CSV or TXT type
<u>FileWriteArray</u>	Writes arrays of any type except for string into a file of BIN type

<u>FileWriteDouble</u>	Writes value of the double type from the current position of a file pointer into a binary file
<u>FileWriteFloat</u>	Writes value of the float type from the current position of a file pointer into a binary file
<u>FileWriteInteger</u>	Writes value of the int type from the current position of a file pointer into a binary file
<u>FileWriteLong</u>	Writes value of the long type from the current position of a file pointer into a binary file
<u>FileWriteString</u>	Writes the value of a string parameter into a BIN or TXT file starting from the current position of the file pointer
<u>FileWriteStruct</u>	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer
<u>FileLoad</u>	Reads all data of a specified binary file into a passed array of numeric types or simple structures
<u>FileSave</u>	Writes to a binary file all elements of an array passed as a parameter
<u>FolderCreate</u>	Creates a folder in the Files directory
<u>FolderDelete</u>	Removes a selected directory. If the folder is not empty, then it can't be removed
<u>FolderClean</u>	Deletes all files in the specified folder

If the file is opened for writing using [FileOpen\(\)](#), all subfolders specified in the path will be created if there are no such ones.

FileFindFirst

The function starts the search of files or subdirectories in a directory in accordance with the specified filter.

```
long FileFindFirst(  
    const string    file_filter,           // String - search filter  
    string&         returned_filename,     // Name of the file or subdirectory found  
    int             common_flag=0         // Defines the search  
);
```

Parameters

file_filter

[in] Search filter. A subdirectory (or sequence of nested subdirectories) relative to the \Files directory, in which files should be searched for, can be specified in the filter.

returned_filename

[out] The returned parameter, where, in case of success, the name of the first found file or subdirectory is placed. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

common_flag

[in] [Flag](#) determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

Returns handle of the object searched, which should be used for further sorting of files and subdirectories by the [FileFindNext\(\)](#) function, or [INVALID_HANDLE](#) when there is no file and subdirectory corresponding to the filter (in the particular case - when the directory is empty). After searching, the handle must be closed using the [FileFindClose\(\)](#) function.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Example:


```

//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="Dir1\\*";
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string file_name;
    string int_dir="";
    int i=1,pos=0,last_pos=-1;
//--- search for the last backslash
    while(!IsStopped())
    {
        pos=StringFind(InpFilter,"\\",pos+1);
        if(pos>=0)
            last_pos=pos;
        else
            break;
    }
//--- the filter contains the folder name
    if(last_pos>=0)
        int_dir=StringSubstr(InpFilter,0,last_pos+1);
//--- get the search handle in the root of the local folder
    long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if the FileFindFirst() is executed successfully
    if(search_handle!=INVALID_HANDLE)
    {
        //--- in a loop, check if the passed strings are the names of files or directories
        do
        {
            ResetLastError();
            //--- if it's a file, the function returns true, and if it's a directory, it
            FileIsExist(int_dir+file_name);
            PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "D":
            i++;
        }
        while(FileFindNext(search_handle,file_name));
        //--- close the search handle
        FileFindClose(search_handle);
    }
    else
        Print("Files not found!");
}

```

See also

[FileFindNext](#), [FileFindClose](#)

FileFindNext

The function continues the search started by [FileFindFirst\(\)](#).

```
bool FileFindNext(
    long      search_handle,      // Search handle
    string&    returned_filename  // Name of the file or subdirectory found
);
```

Parameters

search_handle

[in] Search handle, retrieved by [FileFindFirst\(\)](#).

returned_filename

[out] The name of the next file or subdirectory found. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

Return Value

If successful returns true, otherwise false.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="*";
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string file_name;
    int i=1;
    //--- receive search handle in local folder's root
    long search_handle=FileFindFirst(InpFilter,file_name);
    //--- check if FileFindFirst() function executed successfully
    if(search_handle!=INVALID_HANDLE)
    {
        //--- check if the passed strings are file or directory names in the loop
        do
        {
            ResetLastError();
            //--- if this is a file, the function will return true, if it is a directory,
            FileIsExist(file_name);
            PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "D":
            i++;
        }
        while(FileFindNext(search_handle,file_name));
```



```
//--- close search handle
FileFindClose(search_handle);
}
else
    Print("Files not found!");
}
```

See also

[FileFindFirst](#), [FileFindClose](#)

FileFindClose

The function closes the search handle.

```
void FileFindClose(  
    long search_handle    // Search handle  
);
```

Parameters

search_handle

[in] Search handle, retrieved by [FileFindFirst\(\)](#).

Return Value

No value returned.

Note

Function must be called to free up system resources.

Example:

```
/*--- display the window of input parameters when launching the script  
#property script_show_inputs  
/*--- filter  
input string InpFilter="*";  
//+-----+  
//| Script program start function |  
//+-----+  
void OnStart()  
{  
    string file_name;  
    int i=1;  
    /*--- receive search handle in local folder's root  
    long search_handle=FileFindFirst(InpFilter,file_name);  
    /*--- check if FileFindFirst() function executed successfully  
    if(search_handle!=INVALID_HANDLE)  
    {  
        /*--- check if the passed strings are file or directory names in the loop  
        do  
        {  
            ResetLastError();  
            /*--- if this is a file, the function will return true, if it is a directory,  
            FileIsExist(file_name);  
            PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File"  
            i++;  
        }  
        while(FileFindNext(search_handle,file_name));  
        /*--- close search handle  
        FileFindClose(search_handle);  
    }  
}
```



```
else
    Print("Files not found!");
}
```

See also

[FileFindFirst](#), [FileFindNext](#)

FileIsExist

Checks the existence of a file.

```
bool FileIsExist(
    const string  file_name,      // File name
    int          common_flag=0    // Search area
);
```

Parameters

file_name

[in] The name of the file being checked

common_flag=0

[in] [Flag](#) determining the location of the file. If `common_flag = FILE_COMMON`, then the file is located in a shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the file is located in a local folder.

Return Value

Returns true, if the specified file exists.

Note

Checked file can turn out to be a subdirectory. In this case, `FileIsExist()` function will return false, while error 5018 will be logged in `_LastError` variable - "This is a directory, not a file" (see example for [FileFindFirst](#) function).

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If `common_flag = FILE_COMMON`, then the function looks for the file in a shared folder for all client terminals `\Terminal\Common\Files`, otherwise the function looks for a file in a local folder (`MQL5\Files` or `MQL5\Tester\Files` in the case of testing).

Example:

```
/-- show the window of input parameters when launching the script
#property script_show_inputs
/-- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string  file_name;      // variable for storing file names
    string  filter="*.txt"; // filter for searching the files
    datetime create_date;   // file creation date
    string  files[];        // list of file names
    int     def_size=25;     // array size by default
    int     size=0;         // number of files
```



```
//--- allocate memory for the array
ArrayResize(files,def_size);
//--- receive the search handle in the local folder's root
long search_handle=FileFindFirst(filter,file_name);
//--- check if FileFindFirst() executed successfully
if(search_handle!=INVALID_HANDLE)
{
    //--- searching files in the loop
    do
    {
        files[size]=file_name;
        //--- increase the array size
        size++;
        if(size==def_size)
        {
            def_size+=25;
            ArrayResize(files,def_size);
        }
        //--- reset the error value
        ResetLastError();
        //--- receive the file creation date
        create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
        //--- check if the file is old
        if(create_date<InpFilesDate)
        {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
        }
    }
    while(FileFindNext(search_handle,file_name));
    //--- close the search handle
    FileFindClose(search_handle);
}
else
{
    Print("Files not found!");
    return;
}
//--- check what files have remained
PrintFormat("Results:");
for(int i=0;i<size;i++)
{
    if(FileIsExist(files[i]))
        PrintFormat("%s file exists!",files[i]);
    else
        PrintFormat("%s file deleted!",files[i]);
}
}
```


See also

[FileFindFirst](#)

FileOpen

The function opens the file with the specified name and flag.

```
int FileOpen(
    string  file_name,           // File name
    int     open_flags,         // Combination of flags
    short   delimiter='\t',     // Delimiter
    uint    codepage=CP_ACP     // Code page
);
```

Parameters

file_name

[in] The name of the file can contain subfolders. If the file is opened for writing, these subfolders will be created if there are no such ones.

open_flags

[in] [combination of flags](#) determining the operation mode for the file. The flags are defined as follows:

FILE_READ file is opened for reading

FILE_WRITE file is opened for writing

FILE_BIN binary read-write mode (no conversion from a string and to a string)

FILE_CSV file of csv type (all recorded items are converted to the strings of unicode or ansi type, and are separated by a delimiter)

FILE_TXT a simple text file (the same as csv, but the delimiter is not taken into account)

FILE_ANSI lines of ANSI type (single-byte symbols)

FILE_UNICODE lines of UNICODE type (double-byte characters)

FILE_SHARE_READ shared reading from several programs

FILE_SHARE_WRITE shared writing from several programs

FILE_COMMON location of the file in a shared folder for all client terminals
\\Terminal\\Common\\Files

delimiter='\t'

[in] value to be used as a separator in txt or csv-file. If the csv-file delimiter is not specified, it defaults to a tab. If the txt-file delimiter is not specified, then no separator is used. If the separator is clearly set to 0, then no separator is used.

codepage=CP_ACP

[in] The value of the code page. For the most-used [code pages](#) provide appropriate constants.

Return Value

If a file has been opened successfully, the function returns the file handle, which is then used to access the file data. In case of failure returns [INVALID_HANDLE](#).

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

The file is opened in the folder of the client terminal in the subfolder MQL5\files (or testing_agent_directory\MQL5\files in case of testing). If FILE_COMMON is specified among flags, the file is opened in a shared folder for all MetaTrader 5 client terminals.

"Named pipes" can be opened according to the following rules:

- Pipe name is a string, which should have the following look: "\\servername\pipe\pipename", where servername - server name in the network, while pipename is a pipe name. If the pipes are used on the same computer, the server name can be omitted but a point should be inserted instead of it: "\\.\pipe\pipename". A client trying to connect the pipe should know its name.
- [FileFlush\(\)](#) and [FileSeek\(\)](#) should be called to the beginning of a file between sequential operations of reading from the pipe and writing to it.

A special symbol '\\' is used in shown strings. Therefore, '\\' should be doubled when writing a name in MQL5 application. It means that the above example should have the following look in the code: "\\.\servername\pipe\pipename".

More information about working with named pipes can be found in the article "[Communicating With MetaTrader 5 Using Named Pipes Without Using DLLs](#)".

Example:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- incorrect file opening method
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
string filename=terminal_data_path+"\\MQL5\\Files\\"+"fractals.csv";
int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
if(filehandle<0)
{
Print("Failed to open the file by the absolute path ");
Print("Error code ",GetLastError());
}

//--- correct way of working in the "file sandbox"
ResetLastError();
filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
if(filehandle!=INVALID_HANDLE)
{
FileWrite(filehandle,TimeCurrent(),Symbol(),EnumToString(_Period));
FileClose(filehandle);
Print("FileOpen OK");
}
else Print("Operation FileOpen failed, error ",GetLastError());
//--- another example with the creation of an enclosed directory in MQL5\Files\
string subfolder="Research";
filehandle=FileOpen(subfolder+"\\fractals.txt",FILE_WRITE|FILE_CSV);
if(filehandle!=INVALID_HANDLE)
{
```



```
FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(_Period));
FileClose(filehandle);
Print("The file must be created in the folder "+terminal_data_path+"\\ "+subfolder);
}
else Print("File open failed, error ",GetLastError());
}
```

See also

[Use of a Codepage](#), [FileFindFirst](#), [FolderCreate](#), [File opening flags](#)

FileClose

Close the file previously opened by [FileOpen\(\)](#).

```
void FileClose(
    int file_handle    // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

No value returned.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";    // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- print the path to the file we are going to use
    PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- reset the error value
    ResetLastError();
    //--- open the file for reading (if the file does not exist, the error will occur)
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_TXT|InpEn
    if(file_handle!=INVALID_HANDLE)
    {
        //--- print the file contents
        while(!FileIsEnding(file_handle))
            Print(FileReadString(file_handle));
        //--- close the file
        FileClose(file_handle);
    }
    else
        PrintFormat("Error, code = %d",GetLastError());
}
```


FileCopy

The function copies the original file from a local or shared folder to another file.

```
bool FileCopy(
    const string src_file_name,    // Name of a source file
    int         common_flag,      // Location
    const string dst_file_name,    // Name of the destination file
    int         mode_flags        // Access mode
);
```

Parameters

src_file_name

[in] File name to copy.

common_flag

[in] [Flag](#) determining the location of the file. If `common_flag = FILE_COMMON`, then the file is located in a shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the file is located in a local folder (for example, `common_flag=0`).

dst_file_name

[in] Result file name.

mode_flags

[in] [Access flags](#). The parameter can contain only 2 flags: `FILE_REWRITE` and/or `FILE_COMMON` - other flags are ignored. If the file already exists, and the `FILE_REWRITE` flag hasn't been specified, then the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the `FILE_REWRITE` flag in the `mode_flags` parameter.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpSrc="source.txt";      // source
input string InpDst="destination.txt"; // copy
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+-----+
//| Script program start function |
//+-----+
void OnStart()
```



```

{
//--- display the source contents (it must exist)
if(!FileDisplay(InpSrc))
    return;
//--- check if the copy file already exists (may not be created)
if(!FileDisplay(InpDst))
{
    //--- the copy file does not exist, copying without FILE_REWRITE flag (correct c
    if(FileCopy(InpSrc,0,InpDst,0))
        Print("File is copied!");
    else
        Print("File is not copied!");
}
else
{
    //--- the copy file already exists, try to copy without FILE_REWRITE flag (inco
    if(FileCopy(InpSrc,0,InpDst,0))
        Print("File is copied!");
    else
        Print("File is not copied!");
    //--- InpDst file's contents remains the same
    FileDisplay(InpDst);
    //--- copy once more with FILE_REWRITE flag (correct copying if the file exists)
    if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
        Print("File is copied!");
    else
        Print("File is not copied!");
}
//--- receive InpSrc file copy
FileDisplay(InpDst);
}
//+-----+
//| Read the file contents |
//+-----+
bool FileDisplay(const string file_name)
{
//--- reset the error value
ResetLastError();
//--- open the file
int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
if(file_handle!=INVALID_HANDLE)
{
    //--- display the file contents in the loop
    Print("+-----+");
    PrintFormat("File name = %s",file_name);
    while(!FileIsEnding(file_handle))
        Print(FileReadString(file_handle));
    Print("+-----+");
    //--- close the file

```



```
        FileClose(file_handle);  
        return(true);  
    }  
    ///--- failed to open the file  
    PrintFormat("%s is not opened, error = %d",file_name,GetLastError());  
    return(false);  
}
```


FileDelete

Deletes the specified file in a local folder of the client terminal.

```
bool FileDelete(
    const string file_name,      // File name to delete
    int         common_flag=0    // Location of the file to delete
);
```

Parameters

file_name

[in] File name.

common_flag=0

[in] [Flag](#) determining the file location. If `common_flag = FILE_COMMON`, then the file is located in a shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the file is located in a local folder.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Deletes the specified file from a local folder of the client terminal (`MQL5\Files` or `MQL5\Tester\Files` in case of testing). If `common_flag = FILE_COMMON`, then the function removes the file from the shared folder for all client terminals `\Terminal\Common\Files`.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string file_name;      // variable for storing file names
    string filter="*.txt"; // filter for searching the files
    datetime create_date;  // file creation date
    string files[];        // list of file names
    int def_size=25;       // array size by default
    int size=0;            // number of files
    //--- allocate memory for the array
    ArrayResize(files,def_size);
    //--- receive the search handle in the local folder's root
    long search_handle=FileFindFirst(filter,file_name);
```



```
//--- check if FileFindFirst() executed successfully
if(search_handle!=INVALID_HANDLE)
{
    //--- searching files in the loop
    do
    {
        files[size]=file_name;
        //--- increase the array size
        size++;
        if(size==def_size)
        {
            def_size+=25;
            ArrayResize(files,def_size);
        }
        //--- reset the error value
        ResetLastError();
        //--- receive the file creation date
        create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
        //--- check if the file is old
        if(create_date<InpFilesDate)
        {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
        }
    }
    while(FileFindNext(search_handle,file_name));
    //--- close the search handle
    FileFindClose(search_handle);
}
else
{
    Print("Files not found!");
    return;
}
//--- check what files have remained
PrintFormat("Results:");
for(int i=0;i<size;i++)
{
    if(FileIsExist(files[i]))
        PrintFormat("%s file exists!",files[i]);
    else
        PrintFormat("%s file deleted!",files[i]);
}
}
```


FileMove

Moves a file from a local or shared folder to another folder.

```
bool FileMove(
    const string src_file_name, // File name for the move operation
    int         common_flag,   // Location
    const string dst_file_name, // Name of the destination file
    int         mode_flags     // Access mode
);
```

Parameters

src_file_name

[in] File name to move/rename.

common_flag

[in] [Flag](#) determining the location of the file. If `common_flag = FILE_COMMON`, then the file is located in a shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the file is located in a local folder (`common_flag=0`).

dst_file_name

[in] File name after operation

mode_flags

[in] [Access flags](#). The parameter can contain only 2 flags: `FILE_REWRITE` and/or `FILE_COMMON` - other flags are ignored. If the file already exists and the `FILE_REWRITE` flag isn't specified, the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the `FILE_REWRITE` flag in the `mode_flags` parameter.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpSrcName="data.txt";
input string InpDstName="newdata.txt";
input string InpSrcDirectory="SomeFolder";
input string InpDstDirectory="OtherFolder";
//+-----+
//| Script program start function |
//+-----+
```



```

void OnStart()
{
    string local=TerminalInfoString(TERMINAL_DATA_PATH);
    string common=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
    ///--- receive file paths
    string src_path;
    string dst_path;
    StringConcatenate(src_path,InpSrcDirectory,"/",InpSrcName);
    StringConcatenate(dst_path,InpDstDirectory,"/",InpDstName);
    ///--- check if the source file exists (if not - exit)
    if(FileIsExist(src_path))
        PrintFormat("%s file exists in the %s\\Files\\%s folder",InpSrcName,local,InpSrcName);
    else
    {
        PrintFormat("Error, %s source file not found",InpSrcName);
        return;
    }
    ///--- check if the result file already exists
    if(FileIsExist(dst_path,FILE_COMMON))
    {
        PrintFormat("%s file exists in the %s\\Files\\%s folder",InpDstName,common,InpDstName);
        ///--- file exists, moving should be performed with FILE_REWRITE flag
        ResetLastError();
        if(FileMove(src_path,0,dst_path,FILE_COMMON|FILE_REWRITE))
            PrintFormat("%s file moved",InpSrcName);
        else
            PrintFormat("Error! Code = %d",GetLastError());
    }
    else
    {
        PrintFormat("%s file does not exist in the %s\\Files\\%s folder",InpDstName,common,InpDstName);
        ///--- the file does not exist, moving should be performed without FILE_REWRITE flag
        ResetLastError();
        if(FileMove(src_path,0,dst_path,FILE_COMMON))
            PrintFormat("%s file moved",InpSrcName);
        else
            PrintFormat("Error! Code = %d",GetLastError());
    }
    ///--- the file is moved; let's check it out
    if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
        Print("Success!");
    else
        Print("Error!");
}

```

See also

[FileIsExist](#)

FileFlush

Writes to a disk all data remaining in the input/output file buffer.

```
void FileFlush(
    int file_handle    // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

No value returned.

Note

When writing to a file, the data may be actually found there only after some time. To save the data in the file instantly, use FileFlush() function. If the function is not used, part of the data that has not been stored in the disk yet, will be forcibly written there only when the file is closed using FileClose() function.

The function should be used when written data is of a certain value. It should be kept in mind that frequent function call may affect the program operation speed.

Function FileFlush () must be called between the operations of reading from a file and writing to it.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- file name for writing
input string InpFileName="example.csv"; // file name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- reset error value
    ResetLastError();
    //--- open the file
    int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
    if(file_handle!=INVALID_HANDLE)
    {
        //--- write data to the file
        for(int i=0;i<1000;i++)
        {
            //--- call write function
            FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYMBOL_BID),Sym
            //--- save data on the disk at each 128th iteration
            if((i & 127)==127)
```



```
{
    //--- now, data will be located in the file and will not be lost in case of
    FileFlush(file_handle);
    PrintFormat("i = %d, OK",i);
}
//--- 0.01 second pause
Sleep(10);
}
//--- close the file
FileClose(file_handle);
}
else
    PrintFormat("Error, code = %d",GetLastError());
}
```

See also[FileClose](#)

FileGetInteger

Gets an integer property of a file. There are two variants of the function.

1. Get a property by the handle of a file.

```
long FileGetInteger(  
    int file_handle, // File handle  
    ENUM_FILE_PROPERTY_INTEGER property_id // Property ID  
);
```

2. Get a property by the file name.

```
long FileGetInteger(  
    const string file_name, // File name  
    ENUM_FILE_PROPERTY_INTEGER property_id, // Property ID  
    bool common_folder=false // The file is viewed in a local  
); // or a common folder of all terminals
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

file_name

[in] File name.

property_id

[in] File property ID. The value can be one of the values of the [ENUM_FILE_PROPERTY_INTEGER](#) enumeration. If the second variant of the function is used, you can receive only the values of the [following properties](#): FILE_EXISTS, FILE_CREATE_DATE, FILE_MODIFY_DATE, FILE_ACCESS_DATE and FILE_SIZE.

common_folder=false

[in] Points to the file location. If the parameter is false, terminal data folder is viewed. Otherwise it is assumed that the file is in the shared folder of all terminals \Terminal\Common\Files ([FILE_COMMON](#)).

Return Value

The value of the property. In case of an error, -1 is returned. To get an error code use the [GetLastError\(\)](#) function.

If a folder is specified when getting properties by the name, the function will have error 5018 (ERR_MQL_FILE_IS_DIRECTORY) in any case, though the return value will be correct.

Note

The function always changes the error code. In case of successful completion the error code is reset to NULL.

Example:

```
//--- display the window of input parameters when launching the script
```



```

#property script_show_inputs
//--- input parameters
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string path=InpDirectoryName+"//"+InpFileName;
    long l=0;
//--- open the file
    ResetLastError();
    int handle=FileOpen(path,FILE_READ|FILE_CSV);
    if(handle!=INVALID_HANDLE)
    {
        //--- print all information about the file
        Print(InpFileName," file info:");
        FileInfo(handle,FILE_EXISTS,l,"bool");
        FileInfo(handle,FILE_CREATE_DATE,l,"date");
        FileInfo(handle,FILE_MODIFY_DATE,l,"date");
        FileInfo(handle,FILE_ACCESS_DATE,l,"date");
        FileInfo(handle,FILE_SIZE,l,"other");
        FileInfo(handle,FILE_POSITION,l,"other");
        FileInfo(handle,FILE_END,l,"bool");
        FileInfo(handle,FILE_IS_COMMON,l,"bool");
        FileInfo(handle,FILE_IS_TEXT,l,"bool");
        FileInfo(handle,FILE_IS_BINARY,l,"bool");
        FileInfo(handle,FILE_IS_CSV,l,"bool");
        FileInfo(handle,FILE_IS_ANSI,l,"bool");
        FileInfo(handle,FILE_IS_READABLE,l,"bool");
        FileInfo(handle,FILE_IS_WRITABLE,l,"bool");
        //--- close the file
        FileClose(handle);
    }
    else
        PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetLastError());
}
//+-----+
//| Display the value of the file property |
//+-----+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
{
//--- receive the property value
    ResetLastError();
    if((l=FileGetInteger(handle,id))!=-1)
    {
        //--- the value received, display it in the correct format

```



```
if(!StringCompare(type,"bool"))
    Print(EnumToString(id)," = ",1 ? "true" : "false");
if(!StringCompare(type,"date"))
    Print(EnumToString(id)," = ",(datetime)1);
if(!StringCompare(type,"other"))
    Print(EnumToString(id)," = ",1);
}
else
    Print("Error, Code = ",GetLastError());
}
```

See also

[File Operations](#), [File Properties](#)

FileIsEnding

Defines the end of a file in the process of reading.

```
bool FileIsEnding(
    int file_handle    // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The function returns true if the file end has been reached in the process of reading or moving of the file pointer.

Note

To define the end of the file, the function tries to read the next string from it. If the string does not exist, the function returns true, otherwise it returns false.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";    // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- print the path to the file we are going to use
    PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- reset the error value
    ResetLastError();
    //--- open the file for reading (if the file does not exist, the error will occur)
    int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_TXT|InpEn
    if(file_handle!=INVALID_HANDLE)
    {
        //--- print the file contents
        while(!FileIsEnding(file_handle))
            Print(FileReadString(file_handle));
        //--- close the file
        FileClose(file_handle);
    }
    else
        PrintFormat("Error, code = %d",GetLastError());
```



```
}
```


FileIsLineEnding

Defines the line end in a text file in the process of reading.

```
bool FileIsLineEnding(
    int file_handle    // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

Returns true if in the process of reading txt or csv-file reached the end of the line (the characters CR-LF).

Example (the file obtained during the execution of an example for [FileWriteString](#) function is used here)

```
//+-----+
//|                                     Demo_FileIsLineEnding.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="RSI.csv";    // file name
input string InpDirectoryName="Data";  // directory name
//--- indicator buffers
double open_buff[];
double high_buff[];
double low_buff[];
double close_buff[];
double color_buff[];
//--- overbought variables
int ovb_ind=0;
int ovb_size=0;
datetime ovb_time[];
```



```

//--- oversold variables
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- variables of array sizes by default
    int ovs_def_size=100;
    int ovs_def_size=100;
//--- allocate memory for arrays
    ArrayResize(ovs_time,ovs_def_size);
    ArrayResize(ovs_size,ovs_def_size);
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        double value;
        //--- read data from file
        while(!FileIsEnding(file_handle))
        {
            //--- read the first value in the string
            value=FileReadNumber(file_handle);
            //--- read to different arrays according to the function result
            if(value>=70)
                ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
            else
                ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
        //--- close the file
        FileClose(file_handle);
        PrintFormat("Data is written, %s file is closed",InpFileName);
    }
    else
    {
        PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- binding the arrays
    SetIndexBuffer(0,open_buff,INDICATOR_DATA);
    SetIndexBuffer(1,high_buff,INDICATOR_DATA);
    SetIndexBuffer(2,low_buff,INDICATOR_DATA);
    SetIndexBuffer(3,close_buff,INDICATOR_DATA);
    SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);

```



```

//---- set the indicator values that will not be visible on the chart
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Read the file's string data |
//+-----+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
{
    bool flag=false;
//--- read till the end of the string or of the file is reached
    while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
    {
        //--- shift the position by reading the number
        if(flag)
            FileReadNumber(file_handle);
        //--- store the current date
        arr[size]=FileReadDatetime(file_handle);
        size++;
        //--- increase the array size if necessary
        if(size==def_size)
        {
            def_size+=100;
            ArrayResize(arr,def_size);
        }
        //--- slip past the first iteration
        flag=true;
    }
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);

```



```
//--- the loop for the bars that have not been handled yet
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- 0 by default
    open_buff[i]=0;
    high_buff[i]=0;
    low_buff[i]=0;
    close_buff[i]=0;
    color_buff[i]=0;
    //--- check if any data is still present
    if(ovb_ind<ovb_size)
        for(int j=ovb_ind;j<ovb_size;j++)
        {
            //--- if the dates coincide, the bar is in the overbought area
            if(time[i]==ovb_time[j])
            {
                open_buff[i]=open[i];
                high_buff[i]=high[i];
                low_buff[i]=low[i];
                close_buff[i]=close[i];
                //--- 0 - red color
                color_buff[i]=0;
                //--- increase the counter
                ovb_ind=j+1;
                break;
            }
        }
    //--- check if any data still exists
    if(ovs_ind<ovs_size)
        for(int j=ovs_ind;j<ovs_size;j++)
        {
            //--- if the dates coincide, the bar is in the oversold area
            if(time[i]==ovs_time[j])
            {
                open_buff[i]=open[i];
                high_buff[i]=high[i];
                low_buff[i]=low[i];
                close_buff[i]=close[i];
                //--- 1 - blue color
                color_buff[i]=1;
                //--- increase the counter
                ovs_ind=j+1;
                break;
            }
        }
}
//--- return value of prev_calculated for next call
return(rates_total);
}
```



```
//+-----+
//| ChartEvent event handler |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
                  )
{
    //-- change the indicator width according to the scale
    if (ChartGetInteger(0, CHART_SCALE) > 3)
        PlotIndexSetInteger(0, PLOT_LINE_WIDTH, 2);
    else
        PlotIndexSetInteger(0, PLOT_LINE_WIDTH, 1);
}
```

See also[FileWriteString](#)

FileReadArray

Reads from a file of BIN type arrays of any type except string (may be an array of structures, not containing strings, and dynamic arrays).

```
uint FileReadArray(
    int    file_handle,           // File handle
    void&  array[],              // Array to record
    int    start=0,              // start array position to write
    int    count=WHOLE_ARRAY     // count to read
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

array[]

[out] An array where the data will be loaded.

start=0

[in] Start position to read from the array.

count=WHOLE_ARRAY

[in] Number of elements to read. By default, reads the entire array (count=[WHOLE_ARRAY](#)).

Return Value

Number of elements read.

Note

String array can be read only from the file of TXT type. If necessary, the function tries to increase the size of the array.

Example (the file obtained after execution of the example for [FileWriteArray](#) function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+-----+
//| Structure for storing price data |
//+-----+
struct prices
{
    datetime    date; // date
    double      bid;  // bid price
    double      ask;  // ask price
};
//+-----+
//| Script program start function |
```



```
//+-----+
void OnStart()
{
//--- structure array
    prices arr[];
//--- file path
    string path=InpDirectoryName+"//"+InpFileName;
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
    if(file_handle!=INVALID_HANDLE)
    {
        //--- read all data from the file to the array
        FileReadArray(file_handle,arr);
        //--- receive the array size
        int size=ArraySize(arr);
        //--- print data from the array
        for(int i=0;i<size;i++)
            Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i].ask);
        Print("Total data = ",size);
        //--- close the file
        FileClose(file_handle);
    }
    else
        Print("File open failed, error ",GetLastError());
}
```

See also

[Variables](#), [FileWriteArray](#)

FileReadBool

Reads from the file of CSV type string from the current position to a delimiter (or till the end of the text line) and converts the read string to a bool type value.

```
bool FileReadBool(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

Line read may be set to "true", "false" or the symbolic representation of integers "0" or "1". A nonzero value is converted to a logical true. The function returns the converted value.

Example (the file obtained after executing the example for [FileWrite](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadBool.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 2
//---- plot Label1
#property indicator_label1 "UpSignal"
#property indicator_type1  DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2 "DownSignal"
#property indicator_type2  DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int ind=0; // index
double upbuff[]; // indicator buffers of up arrows
double downbuff[]; // indicator buffer of down arrows
```



```

bool      sign_buff[]; // signal array (true - buy, false - sell)
datetime  time_buff[]; // array of signals' arrival time
int       size=0;      // size of signal arrays
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- open the file
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is open for reading",InpFileName);
    //--- first, read the number of signals
    size=(int)FileReadNumber(file_handle);
    //--- allocate memory for the arrays
    ArrayResize(sign_buff,size);
    ArrayResize(time_buff,size);
    //--- read data from the file
    for(int i=0;i<size;i++)
    {
        //--- signal time
        time_buff[i]=FileReadDatetime(file_handle);
        //--- signal value
        sign_buff[i]=FileReadBool(file_handle);
    }
    //--- close the file
    FileClose(file_handle);
}
else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}

//--- binding the arrays
SetIndexBuffer(0,upbuff,INDICATOR_DATA);
SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
PlotIndexSetInteger(0,PLOT_ARROW,241);
PlotIndexSetInteger(1,PLOT_ARROW,242);
//--- set the indicator values that will not be seen on the chart
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |

```



```
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time, false);
    ArraySetAsSeries(low, false);
    ArraySetAsSeries(high, false);
    //--- the loop for the bars that have not been handled yet
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- 0 by default
        upbuff[i]=0;
        downbuff[i]=0;
        //--- check if any data is still present
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- if dates coincide, use the value from the file
                if(time[i]==time_buff[j])
                {
                    //--- draw the arrow according to the signal
                    if(sign_buff[j])
                        upbuff[i]=high[i];
                    else
                        downbuff[i]=low[i];
                    //--- increase the counter
                    ind=j+1;
                    break;
                }
            }
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
```

See also

[Type bool](#), [FileWrite](#)

FileReadDatetime

Reads from the file of CSV type a string of one of the formats: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" or "HH:MI:SS" - and converts it into a value of datetime type.

```
datetime FileReadDatetime(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of datetime type.

Example (the file obtained after executing the example for [FileWrite](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadDateTime.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 2
//---- plot Label1
#property indicator_label1 "UpSignal"
#property indicator_type1  DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2 "DownSignal"
#property indicator_type2  DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int ind=0; // index
double upbuff[]; // indicator buffers of up arrows
double downbuff[]; // indicator buffer of down arrows
bool sign_buff[]; // signal array (true - buy, false - sell)
```



```

datetime time_buff[]; // array of signals' arrival time
int      size=0;      // size of signal arrays
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- open the file
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is open for reading",InpFileName);
    //--- first, read the number of signals
    size=(int)FileReadNumber(file_handle);
    //--- allocate memory for the arrays
    ArrayResize(sign_buff,size);
    ArrayResize(time_buff,size);
    //--- read data from the file
    for(int i=0;i<size;i++)
    {
        //--- signal time
        time_buff[i]=FileReadDatetime(file_handle);
        //--- signal value
        sign_buff[i]=FileReadBool(file_handle);
    }
    //--- close the file
    FileClose(file_handle);
}
else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}
//--- binding the arrays
SetIndexBuffer(0,upbuff,INDICATOR_DATA);
SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
PlotIndexSetInteger(0,PLOT_ARROW,241);
PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- set the indicator values that will not be seen on the chart
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+

```



```

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time, false);
    ArraySetAsSeries(low, false);
    ArraySetAsSeries(high, false);
    //--- the loop for the bars that have not been handled yet
    for(int i=prev_calculated; i<rates_total; i++)
    {
        //--- 0 by default
        upbuff[i]=0;
        downbuff[i]=0;
        //--- check if any data is still present
        if(ind<size)
        {
            for(int j=ind; j<size; j++)
            {
                //--- if dates coincide, use the value from the file
                if(time[i]==time_buff[j])
                {
                    //--- draw the arrow according to the signal
                    if(sign_buff[j])
                        upbuff[i]=high[i];
                    else
                        downbuff[i]=low[i];
                    //--- increase the counter
                    ind=j+1;
                    break;
                }
            }
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}

```

See also

[Type datetime](#), [StringToTime](#), [TimeToString](#), [FileWrite](#)

FileReadDouble

Reads a double-precision floating point number (double) from the current position of the binary file.

```
double FileReadDouble(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of double type.

Note

For more details about the error, call [GetLastError\(\)](#).

Example (the file obtained after executing the example for [FileWriteDouble](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadDouble.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- data reading parameters
input string InpFileName="MA.csv"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int ind=0;
int size=0;
double ma_buff[];
datetime time_buff[];
//--- indicator buffer
double buff[];
//+-----+
```



```

//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- open the file
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//" +InpFileName,FILE_READ|FILE_BIN);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is available for reading",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- first, read the amount of data in the file
    size=(int)FileReadDouble(file_handle);
    //--- allocate memory for the arrays
    ArrayResize(ma_buff,size);
    ArrayResize(time_buff,size);
    //--- read data from the file
    for(int i=0;i<size;i++)
    {
        time_buff[i]=(datetime)FileReadDouble(file_handle);
        ma_buff[i]=FileReadDouble(file_handle);
    }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}

//--- bind the array to the indicator buffer with index 0
SetIndexBuffer(0,buff,INDICATOR_DATA);
//--- set the indicator values that will not be visible on the chart
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],

```



```
        const long &volume[],
        const int &spread[])
{
    ArraySetAsSeries(time,false);
    ///--- the loop for the bars that have not been handled yet
    for(int i=prev_calculated;i<rates_total;i++)
    {
        ///--- 0 by default
        buff[i]=0;
        ///--- check if any data still exists
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                ///--- if the dates coincide, the value from the file is used
                if(time[i]==time_buff[j])
                {
                    buff[i]=ma_buff[j];
                    ///--- increase the counter
                    ind=j+1;
                    break;
                }
            }
        }
    }
    ///--- return value of prev_calculated for next call
    return(rates_total);
}
```

See also

[Real types \(double, float\)](#), [StringToDouble](#), [DoubleToString](#), [FileWriteDouble](#)

FileReadFloat

Reads the single-precision floating point number (float) from the current position of the binary file.

```
float FileReadFloat(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of float type.

Note

For more details about the error, call [GetLastError\(\)](#).

Example (the file obtained after executing the example for [FileWriteFloat](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadFloat.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "CloseLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="Close.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int ind=0;
int size=0;
double close_buff[];
datetime time_buff[];
//--- indicator buffers
double buff[];
double color_buff[];
//+-----+
```



```

//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    int def_size=100;
//--- allocate memory for the arrays
    ArrayResize(close_buff,def_size);
    ArrayResize(time_buff,def_size);
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_BIN);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- read data from the file
        while(!FileIsEnding(file_handle))
        {
            //--- read time and price values
            time_buff[size]=(datetime)FileReadDouble(file_handle);
            close_buff[size]=(double)FileReadFloat(file_handle);
            size++;
            //--- increase the array sizes if they are overflowed
            if(size==def_size)
            {
                def_size+=100;
                ArrayResize(close_buff,def_size);
                ArrayResize(time_buff,def_size);
            }
        }
//--- close the file
        FileClose(file_handle);
        PrintFormat("Data is read, %s file is closed",InpFileName);
    }
    else
    {
        PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- bind the arrays to the indicator buffers
    SetIndexBuffer(0,buff,INDICATOR_DATA);
    SetIndexBuffer(1,color_buff,INDICATOR_COLOR_INDEX);
//--- set the indicator values that will not be visible on the chart
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |

```



```
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time, false);
    //--- the loop for the bars that have not been handled yet
    for(int i=prev_calculated; i<rates_total; i++)
    {
        //--- 0 by default
        buff[i]=0;
        color_buff[i]=0; // red color by default
        //--- check if any data is still present
        if(ind<size)
        {
            for(int j=ind; j<size; j++)
            {
                //--- if the dates coincide, the value from the file is used
                if(time[i]==time_buff[j])
                {
                    //--- receive the price
                    buff[i]=close_buff[j];
                    //--- if the current price exceeds the previous one, the color is blue
                    if(buff[i-1]>buff[i])
                        color_buff[i]=1;
                    //--- increase the counter
                    ind=j+1;
                    break;
                }
            }
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
```

See also

[Real types \(double, float\)](#), [FileReadDouble](#), [FileWriteFloat](#)

FileReadInteger

The function reads int, short or char value from the current position of the file pointer depending on the length specified in bytes.

```
int FileReadInteger(
    int file_handle,      // File handle
    int size=INT_VALUE    // Size of an integer in bytes
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive) that should be read. The corresponding constants are provided: CHAR_VALUE = 1, SHORT_VALUE = 2 and INT_VALUE = 4, so the function can read the whole value of char, short or int type.

Return Value

A value of the int type. The result of this function must be explicitly cast to a target type, i.e. to the type of data that you need to read. Since a value of the int type is returned, it can be easily converted to any integer value. The file pointer is shifted by the number of bytes read.

Note

When reading less than 4 bytes, the received result is always positive. If one or two bytes are read, the sign of the number can be determined by explicit casting to type char (1 byte) or short (2 bytes). Getting the sign for a three-byte number is not trivial, since there is no corresponding [underlying type](#).

Example (the file obtained after executing the example for [FileWriteInteger](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadInteger.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```



```

//--- parameters for data reading
input string InpFileName="Trend.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;
int      size=0;
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    int def_size=100;
//--- allocate memory for the array
    ArrayResize(time_buff,def_size);
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_BIN);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- additional variables
        int   arr_size;
        uchar arr[];
//--- read data from the file
        while(!FileIsEnding(file_handle))
        {
            //--- find out how many symbols are used for writing the time
            arr_size=FileReadInteger(file_handle,INT_VALUE);
            ArrayResize(arr,arr_size);
            for(int i=0;i<arr_size;i++)
                arr[i]=(char)FileReadInteger(file_handle,CHAR_VALUE);
            //--- store the time value
            time_buff[size]=StringToTime(CharArrayToString(arr));
            size++;
            //--- increase the sizes of the arrays if they are overflowed
            if(size==def_size)
            {
                def_size+=100;
                ArrayResize(time_buff,def_size);
            }
        }
//--- close the file
        FileClose(file_handle);
        PrintFormat("Data is read, %s file is closed",InpFileName);
    }
}

```



```

else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}

//--- bind the array to the indicator buffer
SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will not be visible on the chart
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(close,false);
    //--- the loop for the bars that have not been handled yet
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- 0 by default
        buff[i]=0;
        //--- check if any data is still present
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- if dates coincide, the value from the file is used
                if(time[i]==time_buff[j])
                {
                    //--- receive the price
                    buff[i]=close[i];
                    //--- increase the counter
                    ind=j+1;
                    break;
                }
            }
        }
    }
}

```



```
    }  
    //--- return value of prev_calculated for next call  
    return(rates_total);  
}
```

See also

[IntegerToString](#), [StringToInteger](#), [Integer types](#), [FileWriteInteger](#)

FileReadLong

The function reads an integer of long type (8 bytes) from the current position of the binary file.

```
long FileReadLong(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of long type.

Example (the file obtained during the execution of an example for [FileWriteLong](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadLong.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
#property indicator_plots  1
//--- plot Label1
#property indicator_label1  "Volume"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrYellow
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- parameters for data reading
input string InpFileName="Volume.bin"; // file name
input string InpDirectoryName="Data";  // directory name
//--- global variables
int ind=0;
int size=0;
long volume_buff[];
datetime time_buff[];
//--- indicator buffers
double buff[];
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
```



```

{
//--- open the file
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_BIN);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is open for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- first, read the amount of data in the file
    size=(int)FileReadLong(file_handle);
    //--- allocate memory for the arrays
    ArrayResize(volume_buff,size);
    ArrayResize(time_buff,size);
    //--- read data from the file
    for(int i=0;i<size;i++)
    {
        time_buff[i]=(datetime)FileReadLong(file_handle);
        volume_buff[i]=FileReadLong(file_handle);
    }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is read, %s file is closed",InpFileName);
}
else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}

//--- bind the array to the indicator buffer with 0 index
SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will be visible on the chart
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{

```



```
    ArraySetAsSeries(time, false);  
    //--- the loop for the bars that have not been handled yet  
    for(int i=prev_calculated; i<rates_total; i++)  
    {  
        //--- 0 by default  
        buff[i]=0;  
        //--- check if any data is still present  
        if(ind<size)  
        {  
            for(int j=ind; j<size; j++)  
            {  
                //--- if dates coincide, the value from the file is used  
                if(time[i]==time_buff[j])  
                {  
                    buff[i]=(double)volume_buff[j];  
                    ind=j+1;  
                    break;  
                }  
            }  
        }  
    }  
    //--- return value of prev_calculated for next call  
    return(rates_total);  
}
```

See also

[Integer types](#), [FileReadInteger](#), [FileWriteLong](#)

FileReadNumber

The function reads from the CSV file a string from the current position till a separator (or till the end of a text string) and converts the read string to a value of double type.

```
double FileReadNumber(
    int file_handle // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of double type.

Example (the file obtained during the execution of an example for [FileWriteString](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadNumber.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="RSI.csv"; // file name
input string InpDirectoryName="Data"; // directory name
//--- indicator buffers
double open_buff[];
double high_buff[];
double low_buff[];
double close_buff[];
double color_buff[];
//--- overbought variables
int ovb_ind=0;
int ovb_size=0;
datetime ovb_time[];
```



```

//--- oversold variables
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- variables of array sizes by default
    int ovs_def_size=100;
    int ovs_def_size=100;
//--- allocate memory for arrays
    ArrayResize(ovs_time,ovs_def_size);
    ArrayResize(ovs_size,ovs_def_size);
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        double value;
        //--- read data from file
        while(!FileIsEnding(file_handle))
        {
            //--- read the first value in the string
            value=FileReadNumber(file_handle);
            //--- read to different arrays according to the function result
            if(value>=70)
                ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
            else
                ReadData(file_handle,ovs_size,ovs_def_size);
        }
        //--- close the file
        FileClose(file_handle);
        PrintFormat("Data is written, %s file is closed",InpFileName);
    }
    else
    {
        PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- binding the arrays
    SetIndexBuffer(0,open_buff,INDICATOR_DATA);
    SetIndexBuffer(1,high_buff,INDICATOR_DATA);
    SetIndexBuffer(2,low_buff,INDICATOR_DATA);
    SetIndexBuffer(3,close_buff,INDICATOR_DATA);
    SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);

```



```

//---- set the indicator values that will not be visible on the chart
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Read the file's string data |
//+-----+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
{
    bool flag=false;
//--- read till the end of the string or of the file is reached
    while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
    {
        //--- shift the carriage after reading the number
        if(flag)
            FileReadNumber(file_handle);
        //--- store the current date
        arr[size]=FileReadDatetime(file_handle);
        size++;
        //--- increase the array size if necessary
        if(size==def_size)
        {
            def_size+=100;
            ArrayResize(arr,def_size);
        }
        //--- slip past the first iteration
        flag=true;
    }
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);

```



```
//--- the loop for the bars that have not been handled yet
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- 0 by default
    open_buff[i]=0;
    high_buff[i]=0;
    low_buff[i]=0;
    close_buff[i]=0;
    color_buff[i]=0;
    //--- check if any date is still present
    if(ovb_ind<ovb_size)
        for(int j=ovb_ind;j<ovb_size;j++)
        {
            //--- if the dates coincide, the bar is in the overbought area
            if(time[i]==ovb_time[j])
            {
                open_buff[i]=open[i];
                high_buff[i]=high[i];
                low_buff[i]=low[i];
                close_buff[i]=close[i];
                //--- 0 - red color
                color_buff[i]=0;
                //--- increase the counter
                ovb_ind=j+1;
                break;
            }
        }
    //--- check if any data still exists
    if(ovs_ind<ovs_size)
        for(int j=ovs_ind;j<ovs_size;j++)
        {
            //--- if the dates coincide, the bar is in the oversold area
            if(time[i]==ovs_time[j])
            {
                open_buff[i]=open[i];
                high_buff[i]=high[i];
                low_buff[i]=low[i];
                close_buff[i]=close[i];
                //--- 1 - blue color
                color_buff[i]=1;
                //--- increase the counter
                ovs_ind=j+1;
                break;
            }
        }
}
//--- return value of prev_calculated for next call
return(rates_total);
}
```



```
//+-----+
//| ChartEvent event handler |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
                  )
{
    //-- change the indicator width according to the scale
    if (ChartGetInteger(0, CHART_SCALE) > 3)
        PlotIndexSetInteger(0, PLOT_LINE_WIDTH, 2);
    else
        PlotIndexSetInteger(0, PLOT_LINE_WIDTH, 1);
}
```

See also[FileWriteString](#)

FileReadString

The function reads a string from the current position of a file pointer in a file.

```
string FileReadString(
    int file_handle,      // File handle
    int length=-1         // Length of the string
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

length=-1

[in] Number of characters to read.

Return Value

Line read (string).

Note

When reading from a bin-file. the length of a string to read must be specified. When reading from a txt-file the string length is not required, and the string will be read from the current position to the line feed character "\r\n". When reading from a csv-file, the string length isn't required also, the string will be read from the current position till the nearest delimiter or till the text string end character.

If the file is opened with FILE_ANSI [flag](#), then the line read is converted to Unicode.

Example (the file obtained after executing the example for [FileWriteInteger](#) function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for data reading
input string InpFileName="Trend.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_BIN|FILE_
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        //--- additional variables
        int str_size;
        string str;
```



```
//--- read data from the file
while(!FileIsEnding(file_handle))
{
    //--- find out how many symbols are used for writing the time
    str_size=FileReadInteger(file_handle,INT_VALUE);
    //--- read the string
    str=FileReadString(file_handle,str_size);
    //--- print the string
    PrintFormat(str);
}
//--- close the file
FileClose(file_handle);
PrintFormat("Data is read, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError())
}
```

See also

[String Type](#), [Conversion Functions](#), [FileWriteInteger](#)

FileReadStruct

The function reads contents into a structure passed as a parameter from a binary-file, starting with the current position of the file pointer.

```
uint FileReadStruct(
    int          file_handle,          // file handle
    const void&  struct_object,        // target structure to which the contents are read
    int          size=-1               // structure size in bytes
);
```

Parameters

file_handle

[in] File descriptor of an open bin-file.

struct_object

[out] The object of this structure. The structure should not contain strings, [dynamic arrays](#) or [virtual functions](#).

size=-1

[in] Number of bytes that should be read. If size is not specified or the indicated value is greater than the size of the structure, the exact size of the specified structure is used.

Return Value

If successful the function returns the number of bytes read. File pointer is moved by the same number of bytes.

Example (the file obtained after using the example for [FileWriteStruct](#) function is used here)

```
//+-----+
//|                                     Demo_FileReadStruct.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots  1
//---- plot Label1
#property indicator_label1  "Candles"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
//--- parameters for receiving data
input string InpFileName="EURUSD.txt"; // file name
input string InpDirectoryName="Data";  // directory name
```



```

//+-----+
//| Structure for storing candlestick data |
//+-----+
struct candlesticks
{
    double      open;  // open price
    double      close; // close price
    double      high;  // high price
    double      low;   // low price
    datetime    date;  // date
};

//--- indicator buffers
double      open_buff[];
double      close_buff[];
double      high_buff[];
double      low_buff[];
//--- global variables
candlesticks cand_buff[];
int         size=0;
int         ind=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    int default_size=100;
    ArrayResize(cand_buff,default_size);
//--- open the file
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_BIN|FILE_
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH)
        //--- read data from the file
        while(!FileIsEnding(file_handle))
        {
            //--- write data to the array
            FileReadStruct(file_handle,cand_buff[size]);
            size++;
            //--- check if the array is overflowed
            if(size==default_size)
            {
                //--- increase the array size
                default_size+=100;
                ArrayResize(cand_buff,default_size);
            }
        }
        //--- close the file
    }
}

```



```

        FileClose(file_handle);
        PrintFormat("Data is read, %s file is closed",InpFileName);
    }
else
{
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}

//--- indicator buffers mapping
SetIndexBuffer(0,open_buff,INDICATOR_DATA);
SetIndexBuffer(1,high_buff,INDICATOR_DATA);
SetIndexBuffer(2,low_buff,INDICATOR_DATA);
SetIndexBuffer(3,close_buff,INDICATOR_DATA);

//--- empty value
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);

//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time,false);
    //--- the loop for the candlesticks that have not been handled yet
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- 0 by default
        open_buff[i]=0;
        close_buff[i]=0;
        high_buff[i]=0;
        low_buff[i]=0;
        //--- check if any data is still present
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- if dates coincide, the value from the file is used
                if(time[i]==cand_buff[j].date)
                {

```



```
        open_buff[i]=cand_buff[j].open;
        close_buff[i]=cand_buff[j].close;
        high_buff[i]=cand_buff[j].high;
        low_buff[i]=cand_buff[j].low;
        ///--- increase the counter
        ind=j+1;
        break;
    }
}
}
}
///--- return value of prev_calculated for next call
return(rates_total);
}
```

See also

[Structures and classes](#), [FileWriteStruct](#)

FileSeek

The function moves the position of the file pointer by a specified number of bytes relative to the specified position.

```
bool FileSeek(
    int          file_handle,    // File handle
    long         offset,        // In bytes
    ENUM_FILE_POSITION origin    // Position for reference
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

offset

[in] The shift in bytes (may take a negative value).

origin

[in] The starting point for the displacement. Can be one of values of [ENUM_FILE_POSITION](#).

Return Value

If successful the function returns true, otherwise false. To obtain information about the [error](#) call the [GetLastError\(\)](#) function.

Note

If the execution of the FileSeek() function results in a negative shift (going beyond the "level boundary" of the file), the file pointer will be set to the file beginning.

If a position is set beyond the "right boundary" of the file (larger than the file size), the next writing to the file will be performed not from the end of the file, but from the position set. In this case indefinite values will be written for the previous file end and the position set.

Example:

```
//+-----+
//|                                     Demo_FileSeek.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+-----+
```



```

//| Script program start function |
//+-----+
void OnStart()
{
//--- specify the value of the variable for generating random numbers
    _RandomSeed=GetTickCount();
//--- variables for positions of the strings' start points
    ulong pos[];
    int size;
//--- reset the error value
    ResetLastError();
//--- open the file
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for reading",InpFileName);
        //--- receive start position for each string in the file
        GetStringPositions(file_handle,pos);
        //--- define the number of strings in the file
        size=ArraySize(pos);
        if(!size)
        {
            //--- stop if the file does not have strings
            PrintFormat("%s file is empty!",InpFileName);
            FileClose(file_handle);
            return;
        }
        //--- make a random selection of a string number
        int ind=MathRand()%size;
        //--- shift position to the starting point of the string
        if(FileSeek(file_handle,pos[ind],SEEK_SET)==true)
        {
            //--- read and print the string with ind number
            PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_hanc
        }
        //--- close the file
        FileClose(file_handle);
        PrintFormat("%s file is closed",InpFileName);
    }
    else
        PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
}
//+-----+
//| The function defines starting points for each of the strings in the file and |
//| places them in arr array |
//+-----+
void GetStringPositions(const int handle,ulong &arr[])
{
//--- default array size

```



```
int def_size=127;
//--- allocate memory for the array
ArrayResize(arr,def_size);
//--- string counter
int i=0;
//--- if this is not the file's end, then there is at least one string
if(!FileIsEnding(handle))
{
    arr[i]=FileTell(handle);
    i++;
}
else
    return; // the file is empty, exit
//--- define the shift in bytes depending on encoding
int shift;
if(FileGetInteger(handle,FILE_IS_ANSI))
    shift=1;
else
    shift=2;
//--- go through the strings in the loop
while(1)
{
    //--- read the string
    FileReadString(handle);
    //--- check for the file end
    if(!FileIsEnding(handle))
    {
        //--- store the next string's position
        arr[i]=FileTell(handle)+shift;
        i++;
        //--- increase the size of the array if it is overflown
        if(i==def_size)
        {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
        }
    }
    else
        break; // end of the file, exit
}
//--- define the actual size of the array
ArrayResize(arr,i);
}
```


FileSize

The function returns the file size in bytes.

```
ulong FileSize(  
    int file_handle    // File handle  
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

The value of type int.

Note

To obtain information about the [error](#) call [GetLastError\(\)](#).

Example:

```
/*--- show the window of input parameters when launching the script  
#property script_show_inputs  
/*--- input parameters  
input ulong   InpThresholdSize=20;           // file threshold size in kilobytes  
input string  InpBigFolderName="big";        // folder for large files  
input string  InpSmallFolderName="small";    // folder for small files  
//+-----+  
//| Script program start function |  
//+-----+  
void OnStart()  
{  
    string  file_name;           // variable for storing file names  
    string  filter="*.csv";      // filter for searching the files  
    ulong   file_size=0;         // file size in bytes  
    int     size=0;              // number of files  
    /*--- print the path to the file we are going to work with  
    PrintFormat("Working in %s\\Files\\ folder",TerminalInfoString(TERMINAL_COMMONDATA_  
    /*--- receive the search handle in common folder's root of all terminals  
    long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);  
    /*--- check if FileFindFirst() has been executed successfully  
    if(search_handle!=INVALID_HANDLE)  
    {  
        /*--- move files in the loop according to their size  
        do  
        {  
            /*--- open the file  
            ResetLastError();  
            int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON);  
            if(file_handle!=INVALID_HANDLE)
```



```
{
    //--- receive the file size
    file_size=FileSize(file_handle);
    //--- close the file
    FileClose(file_handle);
}
else
{
    PrintFormat("Failed to open %s file, Error code = %d",file_name,GetLastError());
    continue;
}
//--- print the file size
PrintFormat("Size of %s file is equal to %d bytes",file_name,file_size);
//--- define the path for moving the file
string path;
if(file_size>InpThresholdSize*1024)
    path=InpBigFolderName+"//"+file_name;
else
    path=InpSmallFolderName+"//"+file_name;
//--- move the file
ResetLastError();
if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))
    PrintFormat("%s file is moved",file_name);
else
    PrintFormat("Error, code = %d",GetLastError());
}
while(FileFindNext(search_handle,file_name));
//--- close the search handle
FileFindClose(search_handle);
}
else
    Print("Files not found!");
}
```


FileTell

The file returns the current position of the file pointer of an open file.

```
ulong FileTell(
    int file_handle    // File handle
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

Return Value

Current position of the file descriptor in bytes from the beginning of the file.

Note

To obtain information about the [error](#) call [GetLastError\(\)](#).

Example:

```
//+-----+
//|                                     Demo_FileTell.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- specify the value of the variable for generating random numbers
    _RandomSeed=GetTickCount();
    //--- variables for positions of the strings' start points
    ulong pos[];
    int size;
    //--- reset the error value
    ResetLastError();
    //--- open the file
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_TXT|InpEn
    if(file_handle!=INVALID_HANDLE)
```



```

{
    PrintFormat("%s file is available for reading",InpFileName);
    //--- receive start position for each string in the file
    GetStringPositions(file_handle,pos);
    //--- define the number of strings in the file
    size=ArraySize(pos);
    if(!size)
    {
        //--- stop if the file does not have strings
        PrintFormat("%s file is empty!",InpFileName);
        FileClose(file_handle);
        return;
    }
    //--- make a random selection of a string number
    int ind=MathRand()%size;
    //--- shift position to the starting point of the string
    FileSeek(file_handle,pos[ind],SEEK_SET);
    //--- read and print the string with ind number
    PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_handle))
    //--- close the file
    FileClose(file_handle);
    PrintFormat("%s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError())
}

//+-----+
//| The function defines starting points for each of the strings in the file and |
//| places them in arr array |
//+-----+
void GetStringPositions(const int handle,ulong &arr[])
{
    //--- default array size
    int def_size=127;
    //--- allocate memory for the array
    ArrayResize(arr,def_size);
    //--- string counter
    int i=0;
    //--- if this is not the file's end, then there is at least one string
    if(!FileIsEnding(handle))
    {
        arr[i]=FileTell(handle);
        i++;
    }
    else
        return; // the file is empty, exit
    //--- define the shift in bytes depending on encoding
    int shift;
    if(FileGetInteger(handle,FILE_IS_ANSI))

```



```
    shift=1;
else
    shift=2;
//--- go through the strings in the loop
while(1)
{
    //--- read the string
    FileReadString(handle);
    //--- check for the file end
    if(!FileIsEnding(handle))
    {
        //--- store the next string's position
        arr[i]=FileTell(handle)+shift;
        i++;
        //--- increase the size of the array if it is overflown
        if(i==def_size)
        {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
        }
    }
    else
        break; // end of the file, exit
}
//--- define the actual size of the array
ArrayResize(arr,i);
}
```


FileWrite

The function is intended for writing of data into a CSV file, delimiter being inserted automatically unless it is equal to 0. After writing into the file, the line end character "\r\n" will be added.

```
uint FileWrite(
    int file_handle, // File handle
    ...             // List of recorded parameters
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

...

[in] The list of parameters separated by commas, to write to the file. The number of written parameters can be up to 63.

Return Value

Number of bytes written.

Note

Numbers will be converted into a text at output (see the [Print\(\)](#) function). Data of the double type are output with the accuracy of 16 digits after the decimal point, and the data can be displayed either in traditional or in scientific format - depending on which format will be the most compact. The data of the float type are shown with 5 digits after the decimal point. To output real numbers with different precision or in a clearly specified format, use [DoubleToString\(\)](#).

Numbers of the bool type are displayed as "true" or "false" strings. Numbers of the datetime type are displayed as "YYYY.MM.DD HH:MI:SS".

Example:

```
//+-----+
//|                                     Demo_FileWrite.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string      InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;    // time frame
input int         InpFastEMAPeriod=12;              // fast EMA period
input int         InpSlowEMAPeriod=26;              // slow EMA period
input int         InpSignalPeriod=9;                // difference averaging per
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // price type
```



```

input datetime      InpDateStart=D'2012.01.01 00:00'; // data copying start date
//--- parameters for writing data to file
input string        InpFileName="MACD.csv"; // file name
input string        InpDirectoryName="Data"; // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date_finish; // data copying end date
    bool      sign_buff[]; // signal array (true - buy, false - sell)
    datetime time_buff[]; // array of signals' arrival time
    int       sign_size=0; // signal array size
    double    macd_buff[]; // array of indicator values
    datetime date_buff[]; // array of indicator dates
    int       macd_size=0; // size of indicator arrays
//--- end time is the current time
    date_finish=TimeCurrent();
//--- receive MACD indicator handle
    ResetLastError();
    int macd_handle=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,InpSlowEMAPeriod);
    if(macd_handle==INVALID_HANDLE)
    {
        //--- failed to receive indicator handle
        PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastError());
        return;
    }
//--- being in the loop until the indicator calculates all its values
    while(BarsCalculated(macd_handle)==-1)
        Sleep(10); // pause to allow the indicator to calculate all its values
//--- copy the indicator values for a certain period of time
    ResetLastError();
    if(CopyBuffer(macd_handle,0,InpDateStart,date_finish,macd_buff)==-1)
    {
        PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
        return;
    }
//--- copy the appropriate time for the indicator values
    ResetLastError();
    if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
    {
        PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
        return;
    }
//--- free the memory occupied by the indicator
    IndicatorRelease(macd_handle);
//--- receive the buffer size
    macd_size=ArraySize(macd_buff);
//--- analyze the data and save the indicator signals to the arrays

```



```

ArrayResize(sign_buff,macd_size-1);
ArrayResize(time_buff,macd_size-1);
for(int i=1;i<macd_size;i++)
{
    //--- buy signal
    if(macd_buff[i-1]<0 && macd_buff[i]>=0)
    {
        sign_buff[sign_size]=true;
        time_buff[sign_size]=date_buff[i];
        sign_size++;
    }
    //--- sell signal
    if(macd_buff[i-1]>0 && macd_buff[i]<=0)
    {
        sign_buff[sign_size]=false;
        time_buff[sign_size]=date_buff[i];
        sign_size++;
    }
}
//--- open the file for writing the indicator values (if the file is absent, it will be created)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is available for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- first, write the number of signals
    FileWrite(file_handle,sign_size);
    //--- write the time and values of signals to the file
    for(int i=0;i<sign_size;i++)
        FileWrite(file_handle,time_buff[i],sign_buff[i]);
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
}

```

See also

[Comment](#), [Print](#), [StringFormat](#)

FileWriteArray

The function writes arrays of any type except for string to a BIN file (can be an array of structures not containing strings or dynamic arrays).

```
uint FileWriteArray(
    int          file_handle,      // File handle
    const void&  array[],         // Array
    int          start=0,         // Start index in the array
    int          count=WHOLE_ARRAY // Number of elements
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

array[]

[out] Array for recording.

start=0

[in] Initial index in the array (number of the first recorded element).

count=WHOLE_ARRAY

[in] Number of items to write ([WHOLE_ARRAY](#) means that all items starting with the number start until the end of the array will be written).

Return Value

Number of recorded items.

Note

String array can be recorded in a TXT file. In this case, strings are automatically ended by the line end characters "\r\n". Depending on the file type ANSI or UNICODE, strings are either converted to ansi-encoding or not.

Example:

```
//+-----+
//|                                     Demo_FileWriteArray.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+-----+
//| Structure for storing price data |
//+-----+
```



```

struct prices
{
    datetime      date; // date
    double        bid;  // bid price
    double        ask;  // ask price
};

//--- global variables
int    count=0;
int    size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];

//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
    //--- allocate memory for the array
    ArrayResize(arr,size);
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //--- write the remaining count strings if count<n
    WriteData(count);
}

//+-----+
//| Expert tick function |
//+-----+

void OnTick()
{
    //--- save data to array
    arr[count].date=TimeCurrent();
    arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
    //--- show current data
    Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[count].ask);
    //--- increase the counter
    count++;
    //--- if the array is filled, write data to the file and zero it out
    if(count==size)
    {
        WriteData(size);
        count=0;
    }
}

```



```
//+-----+
//| Write n elements of the array to file |
//+-----+
void WriteData(const int n)
{
    //--- open the file
    ResetLastError();
    int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
    if(handle!=INVALID_HANDLE)
    {
        //--- write array data to the end of the file
        FileSeek(handle,0,SEEK_END);
        FileWriteArray(handle,arr,0,n);
        //--- close the file
        FileClose(handle);
    }
    else
        Print("Failed to open the file, error ",GetLastError());
}
```

See also

[Variables](#), [FileSeek](#)

FileWriteDouble

The function writes the value of a double parameter to a file, starting from the current position of the file pointer.

```
uint FileWriteDouble(
    int     file_handle,    // File handle
    double  value           // Value to write
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

value

[in] The value of double type.

Return Value

If successful the function returns the number of bytes written (in this case [sizeof](#)(double)=8). The file pointer is shifted by the same number of bytes.

Example:

```
//+-----+
//|                                     Demo_FileWriteDouble.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
/-- show the window of input parameters when launching the script
#property script_show_inputs
/-- parameters for receiving data from the terminal
input string      InpSymbolName="EURJPY";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;    // time frame
input int          InpMAPeriod=10;                   // smoothing period
input int          InpMAShift=0;                     // indicator shift
input ENUM_MA_METHOD InpMAMethod=MODE_SMA;           // smoothing type
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // price type
input datetime      InpDateStart=D'2013.01.01 00:00'; // data copying start date
/-- parameters for writing data to the file
input string        InpFileName="MA.csv";           // file name
input string        InpDirectoryName="Data";        // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
```



```

datetime date_finish=TimeCurrent();
double   ma_buff[];
datetime time_buff[];
int       size;
//--- receive MA indicator handle
ResetLastError();
int ma_handle=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,InpMAMethod,
if(ma_handle==INVALID_HANDLE)
{
    //--- failed to receive the indicator handle
    PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastError());
    return;
}
//--- being in the loop until the indicator calculates all its values
while(BarsCalculated(ma_handle)==-1)
    Sleep(20); // a pause to allow the indicator to calculate all its values
PrintFormat("Indicator values starting from %s will be written to the file",TimeToStr(date_finish));
//--- copy the indicator values
ResetLastError();
if(CopyBuffer(ma_handle,0,InpDateStart,date_finish,ma_buff)==-1)
{
    PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError());
    return;
}
//--- copy the time of the appropriate bars' arrival
ResetLastError();
if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
    return;
}
//--- receive the buffer size
size=ArraySize(ma_buff);
//--- free the memory occupied by the indicator
IndicatorRelease(ma_handle);
//--- open the file for writing the indicator values (if the file is absent, it will be created)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is available for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- first, write the size of data sample
    FileWriteDouble(file_handle,(double)size);
    //--- write the indicator time and value to the file
    for(int i=0;i<size;i++)
    {
        FileWriteDouble(file_handle,(double)time_buff[i]);
        FileWriteDouble(file_handle,ma_buff[i]);
    }
}

```



```
    }  
    //--- close the file  
    FileClose(file_handle);  
    PrintFormat("Data is written, %s file is closed",InpFileName);  
    }  
else  
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError())  
}
```

See also

[Real types \(double, float\)](#)

FileWriteFloat

The function writes the value of the float parameter to a bin-file, starting from the current position of the file pointer.

```
uint FileWriteFloat(
    int    file_handle,    // File handle
    float  value           // Value to be written
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

value

[in] The value of float type.

Return Value

If successful the function returns the number of bytes written (in this case [sizeof\(float\)=4](#)). The file pointer is shifted by the same number of bytes.

Example:

```
//+-----+
//|                                     Demo_FileWriteFloat.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
/-- show the window of input parameters when launching the script
#property script_show_inputs
/-- parameters for receiving data from the terminal
input string      InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;    // time frame
input datetime     InpDateStart=D'2013.01.01 00:00'; // data copying start date
/-- parameters for writing data to the file
input string       InpFileName="Close.bin"; // file name
input string       InpDirectoryName="Data"; // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date_finish=TimeCurrent();
    double   close_buff[];
    datetime time_buff[];
    int      size;
```



```

//--- reset the error value
ResetLastError();
//--- copy the close price for each bar
if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1)
{
    PrintFormat("Failed to copy close price values. Error code = %d",GetLastError());
    return;
}
//--- copy the time for each bar
if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Failed to copy the time values. Error code = %d",GetLastError());
    return;
}
//--- receive the buffer size
size=ArraySize(close_buff);
//--- open the file for writing the values (if the file is absent, it will be created)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is open for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- write close prices' time and values to the file
    for(int i=0;i<size;i++)
    {
        FileWriteDouble(file_handle,(double)time_buff[i]);
        FileWriteFloat(file_handle,(float)close_buff[i]);
    }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
}

```

See also

[Real types \(double, float\)](#), [FileWriteDouble](#)

FileWriteInteger

The function writes the value of the int parameter to a bin-file, starting from the current position of the file pointer.

```
uint FileWriteInteger(
    int file_handle,      // File handle
    int value,            // Value to be written
    int size=INT_VALUE    // Size in bytes
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

value

[in] Integer value.

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive), that should be written. The corresponding constants are provided: CHAR_VALUE=1, SHORT_VALUE=2 and INT_VALUE=4, so the function can write the integer value of char, uchar, short, ushort, int, or uint type.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Example:

```
//+-----+
//|                                     Demo_FileWriteInteger.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string      InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;    // time frame
input datetime     InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string       InpFileName="Trend.bin";         // file name
input string       InpDirectoryName="Data";         // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
```



```

{
    datetime date_finish=TimeCurrent();
    double   close_buff[];
    datetime time_buff[];
    int      size;
    //--- reset the error value
    ResetLastError();
    //--- copy the close price for each bar
    if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1)
    {
        PrintFormat("Failed to copy the values of close prices. Error code = %d",GetLastError());
        return;
    }
    //--- copy the time for each bar
    if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
    {
        PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
        return;
    }
    //--- receive the buffer size
    size=ArraySize(close_buff);
    //--- open the file for writing the values (if the file is absent, it will be created)
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is available for writing",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        //---
        int up_down=0; // trend flag
        int arr_size; // arr array size
        uchar arr[]; // uchar type array
        //--- write time values to the file
        for(int i=0;i<size-1;i++)
        {
            //--- compare close prices of the current and next bars
            if(close_buff[i]<=close_buff[i+1])
            {
                if(up_down!=1)
                {
                    //--- write date value to the file using FileWriteInteger
                    StringToCharArray(TimeToString(time_buff[i]),arr);
                    arr_size=ArraySize(arr);
                    //--- first, write the number of symbols in the array
                    FileWriteInteger(file_handle,arr_size,INT_VALUE);
                    //--- write the symbols
                    for(int j=0;j<arr_size;j++)
                        FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
                    //--- change the trend flag

```



```

        up_down=1;
    }
}
else
{
    if(up_down!=-1)
    {
        ///--- write the date value to the file using FileWriteInteger
        StringToCharArray(TimeToString(time_buff[i]),arr);
        arr_size=ArraySize(arr);
        ///--- first, write the number of symbols in the array
        FileWriteInteger(file_handle,arr_size,INT_VALUE);
        ///--- write the symbols
        for(int j=0;j<arr_size;j++)
            FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
        ///--- change the trend flag
        up_down=-1;
    }
}
}
///--- close the file
FileClose(file_handle);
PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError())
}

```

See also

[IntegerToString](#), [StringToInteger](#), [Integer types](#)

FileWriteLong

The function writes the value of the long-type parameter to a bin-file, starting from the current position of the file pointer.

```
uint FileWriteLong(
    int   file_handle,    // File handle
    long  value           // Value to be written
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

value

[in] Value of type long.

Return Value

If successful the function returns the number of bytes written (in this case [sizeof\(long\)](#)=8). The file pointer is shifted by the same number of bytes.

Example:

```
//+-----+
//|                                     Demo_FileWriteLong.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string        InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;      // time frame
input datetime       InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string         InpFileName="Volume.bin"; // file name
input string         InpDirectoryName="Data"; // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date_finish=TimeCurrent();
    long      volume_buff[];
    datetime  time_buff[];
    int       size;
```



```

//--- reset the error value
ResetLastError();
//--- copy tick volumes for each bar
if(CopyTickVolume(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,volume_buff)
{
    PrintFormat("Failed to copy values of the tick volume. Error code = %d",GetLastError());
    return;
}
//--- copy the time for each bar
if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
    return;
}
//--- receive the buffer size
size=ArraySize(volume_buff);
//--- open the file for writing the indicator values (if the file is absent, it will be created)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\ "+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is available for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- first, write the data sample size
    FileWriteLong(file_handle,(long)size);
    //--- write time and volume values to file
    for(int i=0;i<size;i++)
    {
        FileWriteLong(file_handle,(long)time_buff[i]);
        FileWriteLong(file_handle,volume_buff[i]);
    }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
}

```

See also

[Integer types](#), [FileWriteInteger](#)

FileWriteString

The function writes the value of a string-type parameter into a BIN, CSV or TXT file starting from the current position of the file pointer. When writing to a CSV or TXT file: if there is a symbol in the string '\n' (LF) without previous character '\r' (CR), then before '\n' the missing '\r' is added.

```
uint FileWriteString(
    int          file_handle,    // File handle
    const string text_string,    // string to write
    int          length=-1      // number of symbols
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

text_string

[in] String.

length=-1

[in] The number of characters that you want to write. This option is needed for writing a string into a BIN file. If the size is not specified, then the entire string without the trailer 0 is written. If you specify a size smaller than the length of the string, then a part of the string without the trailer 0 is written. If you specify a size greater than the length of the string, the string is filled by the appropriate number of zeros. For files of CSV and TXT type, this parameter is ignored and the string is written entirely.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Note

Note that when writing to a file opened by the FILE_UNICODE [flag](#) (or without a flag FILE_ANSI), then the number of bytes written will be twice as large as the number of string characters written. When recording to a file opened with the FILE_ANSI flag, the number of bytes written will coincide with the number of string characters written.

Example:

```
//+-----+
//|                                     Demo_FileWriteString.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
/-- show the window of input parameters when launching the script
#property script_show_inputs
/-- parameters for receiving data from the terminal
```



```

input string      InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;    // time frame
input int         InpMAPeriod=14;                  // MA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // price type
input datetime    InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string      InpFileName="RSI.csv";           // file name
input string      InpDirectoryName="Data";         // directory name
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date_finish; // data copying end date
    double    rsi_buff[]; // array of indicator values
    datetime  date_buff[]; // array of the indicator dates
    int       rsi_size=0; // size of the indicator arrays
//--- end time is the current one
    date_finish=TimeCurrent();
//--- receive RSI indicator handle
    ResetLastError();
    int rsi_handle=iRSI(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpAppliedPrice);
    if(rsi_handle==INVALID_HANDLE)
    {
        //--- failed to receive the indicator handle
        PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastError());
        return;
    }
//--- being in the loop, until the indicator calculates all its values
    while(BarsCalculated(rsi_handle)==-1)
        Sleep(10); // a pause to allow the indicator to calculate all its values
//--- copy the indicator values for a certain period of time
    ResetLastError();
    if(CopyBuffer(rsi_handle,0,InpDateStart,date_finish,rsi_buff)==-1)
    {
        PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
        return;
    }
//--- copy the appropriate time for the indicator values
    ResetLastError();
    if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
    {
        PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
        return;
    }
//--- free the memory occupied by the indicator
    IndicatorRelease(rsi_handle);
//--- receive the buffer size
    rsi_size=ArraySize(rsi_buff);

```



```

//--- open the file for writing the indicator values (if the file is absent, it will be created)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("%s file is available for writing",InpFileName);
    PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- prepare additional variables
    string str="";
    bool   is_formed=false;
    //--- write dates of forming overbought and oversold areas
    for(int i=0;i<rsi_size;i++)
    {
        //--- check the indicator values
        if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
        {
            //--- if the value is the first one in this area
            if(!is_formed)
            {
                //--- add the value and the date
                str=(string)rsi_buff[i]+"\\t"+(string)date_buff[i];
                is_formed=true;
            }
            else
                str+="\\t"+(string)rsi_buff[i]+"\\t"+(string)date_buff[i];
            //--- move to the next loop iteration
            continue;
        }
        //--- check the flag
        if(is_formed)
        {
            //--- the string is formed, write it to the file
            FileWriteString(file_handle,str+"\\r\\n");
            is_formed=false;
        }
    }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
}
else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
}

```

See also

[String Type](#), [StringFormat](#)

FileWriteStruct

The function writes into a bin-file contents of a structure passed as a parameter, starting from the current position of the file pointer.

```
uint FileWriteStruct(
    int          file_handle,      // File handle
    const void&  struct_object,    // link to an object
    int          size=-1           // size to be written in bytes
);
```

Parameters

file_handle

[in] File descriptor returned by [FileOpen\(\)](#).

struct_object

[in] Reference to the object of this structure. The structure should not contain strings, [dynamic arrays](#) or [virtual functions](#).

size=-1

[in] Number of bytes that you want to record. If size is not specified or the specified number of bytes is greater than the size of the structure, the entire structure is written.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Example:

```
//+-----+
//|                                     Demo_FileWriteStruct.mq5 |
//|                                     Copyright 2013, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string      InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;    // time frame
input datetime     InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string       InpFileName="EURUSD.txt";        // file name
input string       InpDirectoryName="Data";        // directory name
//+-----+
//| Structure for storing candlestick data |
//+-----+
struct candlesticks
```



```

{
    double          open;   // open price
    double          close;  // close price
    double          high;   // high price
    double          low;    // low price
    datetime        date;   // date
};

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime        date_finish=TimeCurrent();
    int             size;
    datetime        time_buff[];
    double          open_buff[];
    double          close_buff[];
    double          high_buff[];
    double          low_buff[];
    candlesticks    cand_buff[];
    //--- reset the error value
    ResetLastError();
    //--- receive the time of the arrival of the bars from the range
    if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
    {
        PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
        return;
    }
    //--- receive high prices of the bars from the range
    if(CopyHigh(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,high_buff)==-1)
    {
        PrintFormat("Failed to copy values of high prices. Error code = %d",GetLastError());
        return;
    }
    //--- receive low prices of the bars from the range
    if(CopyLow(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,low_buff)==-1)
    {
        PrintFormat("Failed to copy values of low prices. Error code = %d",GetLastError());
        return;
    }
    //--- receive open prices of the bars from the range
    if(CopyOpen(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,open_buff)==-1)
    {
        PrintFormat("Failed to copy values of open prices. Error code = %d",GetLastError());
        return;
    }
    //--- receive close prices of the bars from the range
    if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1)
    {

```



```

        PrintFormat("Failed to copy values of close prices. Error code = %d",GetLastError());
        return;
    }
    //--- define dimension of the arrays
    size=ArraySize(time_buff);
    //--- save all data in the structure array
    ArrayResize(cand_buff,size);
    for(int i=0;i<size;i++)
    {
        cand_buff[i].open=open_buff[i];
        cand_buff[i].close=close_buff[i];
        cand_buff[i].high=high_buff[i];
        cand_buff[i].low=low_buff[i];
        cand_buff[i].date=time_buff[i];
    }

    //--- open the file for writing the structure array to the file (if the file is absent)
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("%s file is open for writing",InpFileName);
        PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
        //--- prepare the counter of the number of bytes
        uint counter=0;
        //--- write array values in the loop
        for(int i=0;i<size;i++)
            counter+=FileWriteStruct(file_handle,cand_buff[i]);
        PrintFormat("%d bytes of information is written to %s file",InpFileName,counter);
        PrintFormat("Total number of bytes: %d * %d * %d = %d, %s",size,5,8,size*5*8,size*5*8);
        //--- close the file
        FileClose(file_handle);
        PrintFormat("Data is written, %s file is closed",InpFileName);
    }
    else
        PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
}

```

See also

[Structures and classes](#)

FileLoad

Reads all data of a specified binary file into a passed array of numeric types or simple structures. The function allows you to quickly read data of a known type into the appropriate array.

```
bool FileLoad(
    int          file_name,           // File name
    const void&  buffer[],           // An array of numeric types or simple structures
    int          common_flag=0       // A file flag, is searched in <data_folder>\MQL5\
);
```

Parameters

file_name

[in] The name of the file from which data will be read.

buffer

[out] An array of numeric types or [simple structures](#).

common_flag=0

[in] [A file flag](#) indicating the operation mode. If the parameter is not specified, the file is searched in the subfolder MQL5\Files (or in <testing_agent_directory>\MQL5\Files in case of testing).

Return Value

The number of elements read or -1 in case of an error.

Note

The FileLoad() function reads from a file the number of bytes multiple of the array element size. Suppose the file size is 10 bytes, and the function reads data into an array of type double ([sizeof](#)(double)=8). In this case the function will read only 8 bytes, the remaining 2 bytes at the end of the file will be dropped, and the function FileLoad() will return 1 (1 element read).

Example:

```
//+-----+
//|                                     Demo_FileLoad.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int      bars_to_save=10; // Number of bars
//+-----+
```



```

///| Script program start function |
//+-----+
void OnStart()
{
    string filename=_Symbol+"_rates.bin";
    MqlRates rates[];
//---
    int copied=CopyRates(_Symbol,_Period,0,bars_to_save,rates);
    if(copied!=-1)
    {
        PrintFormat(" CopyRates(%s) copied %d bars",_Symbol,copied);
        //--- Writing quotes to a file
        if(!FileSave(filename,rates,FILE_COMMON))
            PrintFormat("FileSave() failed, error=%d",GetLastError());
    }
    else
        PrintFormat("Failed CopyRates(%s), error=",_Symbol,GetLastError());
//--- Now reading these quotes back to the file
    ArrayFree(rates);
    long count=FileLoad(filename,rates,FILE_COMMON);
    if(count!=-1)
    {
        Print("Time\tOpen\tHigh\tLow\tClose\tTick Voulme\tSpread\tReal Volume");
        for(int i=0;i<count;i++)
        {
            PrintFormat("%s\t%G\t%G\t%G\t%G\t%I64u\t%d\t%I64u",
                TimeToString(rates[i].time,TIME_DATE|TIME_SECONDS),
                rates[i].open,rates[i].high,rates[i].low,rates[i].close,
                rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
    }
}

```

See also

[Structures and Classes](#), [FileReadArra](#), [FileReadStruct](#)

FileSave

Writes to a binary file all elements of an array passed as a parameter. The function allows you to quickly write arrays of numeric types or simple structures as one string.

```
bool FileSave(
    int          file_name,           // File name
    const void&  buffer[],           // An array of numeric types or simple structures
    int          common_flag=0       // A file flag, by default files are written to <default_directory>
);
```

Parameters

file_name

[in] The name of the file, to the data array will be written.

buffer

[in] An array of numeric types or [simple structures](#).

common_flag=0

[in] [A file flag](#) indicating the operation mode. If the parameter is not specified, the file will be written to the subfolder MQL5\Files (or to <testing_agent_directory>\MQL5\Files in case of testing).

Return Value

In case of failure returns false.

Example:

```
//+-----+
//|                                     Demo_FileSave.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int          ticks_to_save=1000; // Number of ticks
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    string filename=_Symbol+"_ticks.bin";
    MqlTick ticks[];
u//---
    int copied=CopyTicks(_Symbol,ticks,COPY_TICKS_ALL,0,ticks_to_save);
    if(copied!=-1)
    {
```



```

PrintFormat(" CopyTicks(%s) copied %d ticks",_Symbol,copied);
//--- If the tick history is synchronized, the error code is equal to zero
if(!GetLastError()==0)
    PrintFormat("%s: Ticks are not synchronized, error=%d",_Symbol,copied,_LastE
//--- Writing ticks to a file
if(!FileSave(filename,ticks,FILE_COMMON))
    PrintFormat("FileSave() failed, error=%d",GetLastError());
}
else
    PrintFormat("Failed CopyTicks(%s), Error=",_Symbol,GetLastError());
//--- Now reading the ticks back to the file
ArrayFree(ticks);
long count=FileLoad(filename,ticks,FILE_COMMON);
if(count!=-1)
{
    Print("Time\tBid\tAsk\tLast\tVolume\tms\tflags");
    for(int i=0;i<count;i++)
    {
        PrintFormat("%s.%03I64u:\tG\tG\tG\tG\tI64u\t0x%04x",
            TimeToString(ticks[i].time,TIME_DATE|TIME_SECONDS),ticks[i].time_msc%1000,
            ticks[i].bid,ticks[i].ask,ticks[i].last,ticks[i].volume,ticks[i].flags);
    }
}
}

```

See also

[Structures and Classes](#), [FileWriteArray](#), [FileWriteStruct](#)

FolderCreate

The function creates a folder in the Files directory (depending on the value of `common_flag`).

```
bool FolderCreate(
    string  folder_name,      // String with the name of the new folder
    int     common_flag=0     // Scope
);
```

Parameters

folder_name

[in] The name of the directory you want to create. Contains the full path to the folder.

common_flag=0

[in] [Flag](#) determining the location of the directory. If `common_flag=FILE_COMMON`, then the directory is in the shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the directory is in a local folder (`MQL5\Files` or `MQL5\Tester\Files` in case of testing).

Return Value

Returns true if successful, otherwise - false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Example:

```
//+-----+
//|                                     Demo_FolderCreate.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderCreate()."
#property description "An external parameter defines the folder for creating folders."
#property description "After running the script, a structure of folders is created"

//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- The input parameter defines the folder, in which the script is running
input bool    common_folder=false; // A shared folder of all terminals
int         flag=0;                // The flag value determines the place for running
//+-----+
//| Script program start function |
//+-----+
void OnStart()
```



```

{
    string working_folder;
    //--- Set the flag value, if the external parameter common_folder==true
    if(common_folder)
    {
        flag=FILE_COMMON;
        //--- Find the folder, in which we are working
        working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL5\\Files";
    }
    else working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
    //--- The folder that will be created in the folder MQL5\\Files
    string root="Folder_A";
    if(CreateFolder(working_folder,root,flag))
    {
        //--- Create a child folder in it Child_Folder_B1
        string folder_B1="Child_Folder_B1";
        string path=root+"\\ "+folder_B1;          // Create a folder name based on the s
        if(CreateFolder(working_folder,path,flag))
        {
            //--- Create 3 more child folders in this folder
            string folder_C11="Child_Folder_C11";
            string child_path=path+"\\ "+folder_C11; // Create a folder name based on the s
            CreateFolder(working_folder,child_path,flag);
            //--- The second child folder
            string folder_C12="Child_Folder_C12";
            child_path=path+"\\ "+folder_C12;
            CreateFolder(working_folder,child_path,flag);

            //--- The third child folder
            string folder_C13="Child_Folder_C13";
            child_path=path+"\\ "+folder_C13;
            CreateFolder(working_folder,child_path,flag);
        }
    }
}
//---
}
//+-----+
//| Tries to create a folder and shows a message |
//+-----+
bool CreateFolder(string working_folder,string folder_path,int file_flag)
{
    //--- A debug message
    PrintFormat("folder_path=%s",folder_path);
    //--- Trying to create a folder relative to path MQL5\\Files
    if(FolderCreate(folder_path,file_flag))
    {
        //--- Show the entire path to the created folder
        PrintFormat("Folder %s has been created",working_folder+"\\ "+folder_path);
        //--- Reset the error code
    }
}

```



```
ResetLastError();  
//--- Return successful result  
return true;  
}  
else  
    PrintFormat("Failed to create folder %s. Error code %d",working_folder+folder_pa  
//--- Failed  
return false;  
}
```

See also

[FileOpen\(\)](#), [FolderClean\(\)](#), [FileCopy\(\)](#)

FolderDelete

The function removes the specified directory. If the folder is not empty, then it can't be removed.

```
bool FolderDelete(
    string  folder_name,      // String with the name of the folder to delete
    int     common_flag=0     // Scope
);
```

Parameters

folder_name

[in] The name of the directory you want to delete. Contains the full path to the folder.

common_flag=0

[in] [Flag](#) determining the location of the directory. If `common_flag=FILE_COMMON`, then the directory is in the shared folder for all client terminals `\Terminal\Common\Files`. Otherwise, the directory is in a local folder (`MQL5\Files` or `MQL5\Tester\Files` in the case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the directory contains at least one file and/or subdirectory, then this directory can't be deleted, it must be cleared first. [FolderClean\(\)](#) is used to clear a folder of all its files or subfolders.

Example:

```
//+-----+
//|                                     Demo_FolderDelete.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
/-- Description
#property description "The script shows a sample use of FolderDelete()."
#property description "First two folders are created; one of them is empty, the second is not."
#property description "When trying to delete a non-empty folder, an error is returned."

/-- Show the dialog of input parameters when starting the script
#property script_show_inputs
/-- Input parameters
input string  firstFolder="empty";      // An empty folder
input string  secondFolder="nonempty";  // The folder, in which one file will be created
string filename="delete_me.txt";       // The name of the file that will be created in the folder
//+-----+
```



```

//| Script program start function |
//+-----+
void OnStart()
{
//--- Write the file handle here
    int handle;
//--- Find out in what folder we are working
    string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- A debug message
    PrintFormat("working_folder=%s",working_folder);
//--- Trying to create an empty folder relative to path MQL5\\Files
    if(FolderCreate(firstFolder,0)) // 0 means that we are working in the local folder
    {
        //--- Enter the full path to the created folder
        PrintFormat("Folder %s has been created",working_folder+"\\ "+firstFolder);
        //--- Reset the error code
        ResetLastError();
    }
    else
        PrintFormat("Failed to create folder %s. Error code %d",working_folder+"\\ "+firstFolder,GetLastError());

//--- Now create a non-empty folder using the FileOpen() function
    string filepath=secondFolder+"\\ "+filename; // Form path to file that we want to create
    handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // Flag FILE_WRITE in this case is correct
    if(handle!=INVALID_HANDLE)
        PrintFormat("File %s has been opened for reading",working_folder+"\\ "+filepath);
    else
        PrintFormat("Failed to create file %s in folder %s. Error code=",filename,secondFolder,GetLastError());

    Comment(StringFormat("Prepare to delete folders %s and %s", firstFolder, secondFolder));
//--- A small pause of 5 seconds to read a message in the chart
    Sleep(5000); // Sleep() cannot be used in indicators!

//--- Show a dialog and ask the user
    int choice=MessageBox(StringFormat("Do you want to delete folders %s and %s?", firstFolder, secondFolder),
        "Deleting folders",
        MB_YESNO|MB_ICONQUESTION); // Two buttons - "Yes" and "No"

//--- Run an action depending on the selected variant
    if(choice==IDYES)
    {
        //--- Delete the comment form the chart
        Comment("");
        //--- Add a message into the "Experts" journal
        PrintFormat("Trying to delete folders %s and %s",firstFolder, secondFolder);
        ResetLastError();
        //--- Delete the empty folder
        if(FolderDelete(firstFolder))
            //--- The following message should appear since the folder is empty
            PrintFormat("Folder %s has been successfully deleted",firstFolder);
    }
}

```



```
    else
        PrintFormat("Failed to delete folder %s. Error code=%d", firstFolder, GetLastError());

    ResetLastError();

    ///--- Delete the folder that contains a file
    if(FolderDelete(secondFolder))
        PrintFormat("Folder %s has been successfully deleted", secondFolder);
    else
        ///--- The following message should appear since the folder contains a file
        PrintFormat("Failed to delete folder %s. Error code=%d", secondFolder, GetLastError());
}
else
    Print("Deletion canceled");
///---
}
```

See also

[FileOpen\(\)](#), [FolderClean\(\)](#), [FileMove\(\)](#)

FolderClean

The function deletes all files in a specified folder.

```
bool FolderClean(
    string folder_name,      // String with the name of the deleted folder
    int common_flag=0       // Scope
);
```

Parameters

folder_name

[in] The name of the directory where you want to delete all files. Contains the full path to the folder.

common_flag=0

[in] [Flag](#) determining the location of the directory. If common_flag = FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder(MQL5\Files or MQL5\Tester\Files in case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

This function should be used with caution, since all the files and all subdirectories are deleted irretrievably.

Example:

```
//+-----+
//|                                     Demo_FolderClean.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderClean()."
#property description "First, files are created in the specified folder using the File
#property description "Then, before the files are deleted, a warning is shown using Me

//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- Input parameters
input string foldername="demo_folder"; // Create a folder in MQL5/Files/
input int    files=5;                  // The number of files to create and delete
//+-----+
```



```

//| Script program start function |
//+-----+
void OnStart()
{
    string name="testfile";
//--- First open or create files in the terminal data folder
    for(int N=0;N<files;N++)
    {
        //--- The name of the file in the form of 'demo_folder\testfileN.txt'
        string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
        //--- Open a file with the flag for writing, in this case the 'demo_folder' will
        int handle=FileOpen(filemane,FILE_WRITE);
        //--- Find out if the FileOpen() function was successful
        if(handle==INVALID_HANDLE)
        {
            PrintFormat("Failed to create file %s. Error code",filemane,GetLastError());
            ResetLastError();
        }
        else
        {
            PrintFormat("File %s has been successfully opened",filemane);
            //--- The opened file is not needed any more, so close it
            FileClose(handle);
        }
    }

//--- Check the number of files in the folder
    int k=FilesInFolder(foldername+"\\*.txt",0);
    PrintFormat("Totally the folder %s contains %d files",foldername,k);

//--- Show a dialog to ask the user
    int choice=MessageBox(StringFormat("You are going to delete %d files from folder %s",k,
        foldername),
        "Deleting files from the folder",
        MB_YESNO|MB_ICONQUESTION); // Two buttons - "Yes" and "No"

    ResetLastError();

//--- Run an action depending on the selected variant
    if(choice==IDYES)
    {
        //--- Start to delete files
        PrintFormat("Trying to delete all files from folder %s",foldername);
        if(FolderClean(foldername,0))
            PrintFormat("Files have been successfully deleted, %d files left in folder %s",
                files,
                FilesInFolder(foldername+"\\*.txt",0));
        else
            PrintFormat("Failed to delete files from folder %s. Error code %d",foldername,GetLastError());
    }
    else
        PrintFormat("Deletion canceled");
}

```



```

//---
}
//+-----+
//| Returns the number of files in the specified folder |
//+-----+
int FilesInFolder(string path,int flag)
{
    int count=0;
    long handle;
    string filename;
//---
    handle=FileFindFirst(path,filename,flag);
//--- If at least one file found, search for more files
    if(handle!=INVALID_HANDLE)
    {
        //--- Show the name of the file
        PrintFormat("File %s found",filename);
        //--- Increase the counter of found files/folders
        count++;
        //--- Start search in all files/folders
        while(FileFindNext(handle,filename))
        {
            PrintFormat("File %s found",filename);
            count++;
        }
        //--- Do not forget to close the search handle upon completion
        FileFindClose(handle);
    }
    else // Failed to get the handle
    {
        PrintFormat("Files search in folder %s failed",path);
    }
//--- Return the result
    return count;
}

```

See also

[FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

Custom Indicators

This is the group functions used in the creation of custom indicators. These functions can't be used when writing Expert Advisors and Scripts.

Function	Action
SetIndexBuffer	Binds the specified indicator buffer with one-dimensional dynamic array of the double type
IndicatorSetDouble	Sets the value of an indicator property of the double type
IndicatorSetInteger	Sets the value of an indicator property of the int type
IndicatorSetString	Sets the value of an indicator property of the string type
PlotIndexSetDouble	Sets the value of an indicator line property of the type double
PlotIndexSetInteger	Sets the value of an indicator line property of the int type
PlotIndexSetString	Sets the value of an indicator line property of the string type
PlotIndexGetInteger	Returns the value of an indicator line property of the integer type

[Indicator properties](#) can be set using the compiler directives or using functions. To better understand this, it is recommended that you study [indicator styles in examples](#).

All the necessary calculations of a custom indicator must be placed in the predetermined function [OnCalculate\(\)](#). If you use a short form of the OnCalculate() function call, like

```
int OnCalculate (const int rates_total, const int prev_calculated, const int begin, co
```

then the *rates_total* variable contains the value of the total number of elements of the price[] array, passed as an input parameter for calculating indicator values.

Parameter *prev_calculated* is the result of the execution of OnCalculate() at the previous call; it allows organizing a saving algorithm for calculating indicator values. For example, if the current value *rates_total* = 1000, *prev_calculated* = 999, then perhaps it's enough to make calculations only for one value of each indicator buffer.

If the information about the size of the input array price would have been unavailable, then it would lead to the necessity to make calculations for 1000 values of each indicator buffer. At the first call of OnCalculate() value *prev_calculated* = 0. If the price[] array has changed somehow, then in this case *prev_calculated* is also equal to 0.

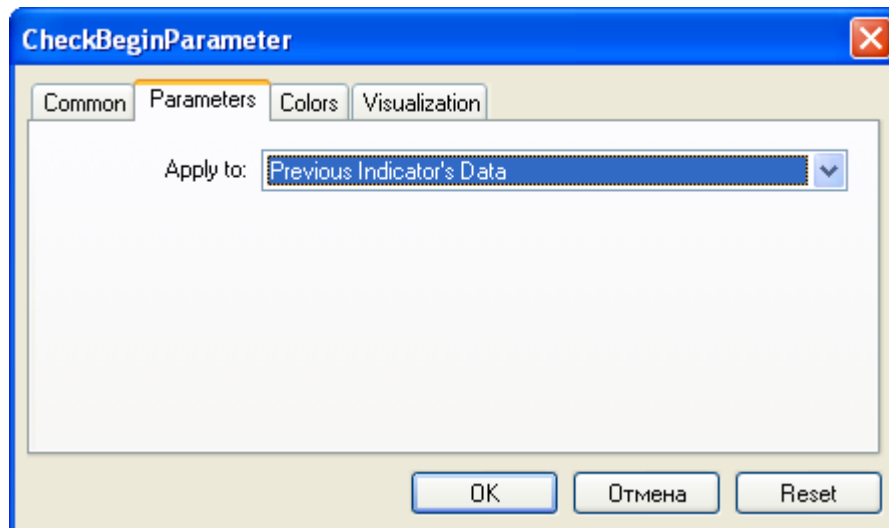
The *begin* parameter shows the number of initial values of the price array, which don't contain data for calculation. For example, if values of Accelerator Oscillator (for which the first 37 values aren't

calculated) were used as an input parameter, then begin = 37. For example, let's consider a simple indicator:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double Label1Buffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const int begin,
               const double &price[])

{
//---
Print("begin = ",begin," prev_calculated = ",prev_calculated," rates_total = ",rates_total);
//--- return value of prev_calculated for next call
return(rates_total);
}
```

Drag it from the "Navigator" window to the window of the Accelerator Oscillator indicator and we indicate that calculations will be made based on the values of the previous indicator:



As a result, the first call of `OnCalculate()` the value of `prev_calculated` will be equal to zero, and in further calls it will be equal to the `rates_total` value (until the number of bars on the price chart increases).



The value of the `begin` parameter will be exactly equal to the number of initial bars, for which the values of the Accelerator indicator aren't calculated according to the logic of this indicator. If we look at the source code of the custom indicator `Accelerator.mq5`, we'll see the following lines in the `OnInit()` function:

```
//--- sets first bar from which index will be drawn
PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, 37);
```


Using the function [PlotIndexSetInteger](#)(0, [PLOT_DRAW_BEGIN](#), empty_first_values), we set the number of non-existing first values in the zero indicator array of a custom indicator, which we don't need to accept for calculation (empty_first_values). Thus, we have mechanisms to:

1. set the number of initial values of an indicator, which shouldn't be used for calculations in another custom indicator;
2. get information on the number of first values to be ignored when you call another custom indicator, without going into the logic of its calculations.

Indicator Styles in Examples

The MetaTrader 5 Client Terminal includes 38 technical indicators that can be used in MQL5 programs using [appropriate functions](#). But the main advantage of the MQL5 language is the ability to create custom indicators, which can then be used in Expert Advisors or simply applied on price charts for the purpose of technical analysis.

The entire set of indicators can be derived from several base [drawing styles](#), known as plotting. Plotting denotes a way of displaying data, which the indicator calculates, stores and provides on request. There are seven such basic plotting types:

1. A line
2. A section (segment)
3. Histogram
4. Arrow (symbol)
5. A painted area (filled channel)
6. Bars
7. Japanese candlesticks

Each plotting requires one to five [arrays](#) of the [double](#) type, in which indicator values are stored. For the purpose of convenience, these arrays are associated with the indicator buffers. The number of buffers in an indicator must be declared in advance using [compiler directives](#), for example:

```
#property indicator_buffers 3 // Number of buffers
#property indicator_plots 2 // number of plots
```

The number of buffers in the indicator is always greater than or equal to the number of plots in the indicator.

Since each basic plotting type can have color variation or construction specifics, the actual number of plotting types in the MQL5 is 18:

Plotting	Description	Value buffers	Color buffers
DRAW_NONE	Is not visually displayed in the chart, but the values of the corresponding buffer can be viewed in the Data Window	1	-
DRAW_LINE	A line is plotted on the values of the corresponding buffer (empty values in the buffer are undesirable)	1	-
DRAW_SECTION	Is drawn as line segments between the values of the corresponding buffer	1	-

	(usually has a lot of empty values)		
<u>DRAW_HISTOGRAM</u>	Is drawn as a histogram from the zero line to the values of the corresponding buffer (may have empty values)	1	-
<u>DRAW_HISTOGRAM2</u>	Is drawn as a histogram based on two indicator buffers (may have empty values)	2	-
<u>DRAW_ARROW</u>	Is drawn as symbols (may have empty values)	1	-
<u>DRAW_ZIGZAG</u>	Similar to the style <u>DRAW_SECTION</u> , but unlike it, can plot vertical segments on one bar	2	-
<u>DRAW_FILLING</u>	Color fill between two lines. 2 values of the corresponding buffers are shown in the Data Window	2	-
<u>DRAW_BARS</u>	Is drawn as bars. 4 values of the corresponding buffers are shown in the Data Window	4	-
<u>DRAW_CANDLES</u>	Drawn as Japanese candlesticks. 4 values of the corresponding buffers are shown in the Data Window	4	-
<u>DRAW_COLOR_LINE</u>	A line for which you can alternate colors on different bars or change its color at any time	1	1
<u>DRAW_COLOR_SECTION</u>	Similar to the style <u>DRAW_SECTION</u> , but the color of each section can be set	1	1

	individually; color can also be set dynamically		
<u>DRAW_COLOR_HISTOGRAM</u>	Similar to the style <u>DRAW_HISTOGRAM</u> , but each strip may have a different color, you can set the color dynamically	1	1
<u>DRAW_COLOR_HISTOGRAM2</u>	Similar to the style <u>DRAW_HISTOGRAM2</u> , but each strip may have a different color, you can set the color dynamically	2	1
<u>DRAW_COLOR_ARROW</u>	Similar to the style <u>DRAW_ARROW</u> , but each symbol can have its color. Color can be changed dynamically	1	1
<u>DRAW_COLOR_ZIGZAG</u>	The <u>DRAW_ZIGZAG</u> style with the options of individual coloring of sections and dynamic color changing	2	1
<u>DRAW_COLOR_BARS</u>	The <u>DRAW_BARS</u> style with the options of individual coloring of bars and dynamic color changing	4	1
<u>DRAW_COLOR_CANDLES</u>	The <u>DRAW_CANDLES</u> style with the options of individual coloring of candlesticks and dynamic color changing	4	1

The difference between an indicator buffer and an array

In each indicator, on its [global level](#), you should declare one or more arrays of the double type, which then must be used as an indicator buffer using the [SetIndexBuffer\(\)](#) function. To draw indicator plots, only the values of the indicator buffers are used, any other arrays cannot be used for this purpose. In addition, buffer values are displayed in the Data Window.

An indicator buffer should be [dynamic](#) and does not require [specification of the size](#) - the size of the array used as the indicator buffer is set by the terminal execution subsystem automatically.

After the array is bound to the indicator buffer, [the indexing direction](#) is set by default like in ordinary arrays, but you can use the [ArraySetAsSeries\(\)](#) function to change the way of access to the array elements. By default, the indicator buffer is used to store data used for plotting ([INDICATOR_DATA](#)).

If the calculation of indicator values requires holding intermediate calculations and storing the additional values for each bar, then such an array can be declared as a calculation buffer during binding ([INDICATOR_CALCULATIONS](#)). For the intermediate values, you can also use a regular array, but in this case, the programmer has to manage the size of the array.

Some plots allow setting a color for each bar. To store the information about color, color buffers are used ([INDICATOR_COLOR_INDEX](#)). The color is an integer type [color](#), but all indicator buffers must be of type [double](#). Values of color and auxiliary ([INDICATOR_CALCULATIONS](#)) buffers cannot be obtained by using [CopyBuffer\(\)](#).

The number of indicator buffers must be specified using the compiler directive `#property indicator_buffers` number_of_buffers:

```
#property indicator_buffers 3 // the indicator has 3 buffers
```

The maximum allowed number of buffers in one indicator is 512.

Relevance of Indicator Buffers and Plotting

Each plotting is based on one or more indicator buffers. So, for displaying simple candlesticks, four values are required - Open, High, Low and Close prices. Accordingly, to display an indicator in the form of candlesticks, it is necessary to declare 4 indicator buffers and 4 arrays of the double type for them. For example:

```
//--- The indicator has four indicator buffers
#property indicator_buffers 4
//--- The indicator has one plotting
#property indicator_plots 1
//--- Graphical plotting number 1 will appear as candlesticks
#property indicator_type1 DRAW_CANDLES
//--- Candlestick will be drawn in clrDodgerBlue
#property indicator_color1 clrDodgerBlue
//--- 4 arrays for the indicator buffers
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

Graphical plots automatically use indicator buffers in accordance with the plot number. Numbering of plots starts with 1, numbering of buffers starts with zero. If the first plotting requires 4 indicator buffers, then the first 4 indicator buffers will be used to draw it. These four buffers should be linked with the appropriate arrays with correct indexing using the [SetIndexBuffer\(\)](#) function.

```
//--- Binding arrays with indicator buffers
SetIndexBuffer(0,OBuffer,INDICATOR_DATA); // The first buffer corresponds to the O
```



```
SetIndexBuffer(1,HBuffer,INDICATOR_DATA); // The second buffer corresponds to inde
SetIndexBuffer(2,LBuffer,INDICATOR_DATA); // The third buffer corresponds to inde
SetIndexBuffer(3,CBuffer,INDICATOR_DATA); // The fourth buffer corresponds to inde
```

The plotting candlesticks, the indicator will use just the first four buffers, because plotting of "candlesticks" was announced under the first number.

Change the example, and add plotting of a simple line - [DRAW_LINE](#). Now suppose that the line is numbered 1, and the candlesticks are number 2. The number of buffers and the number of plots has increased.

```
//--- The indicator has 5 indicator buffers
#property indicator_buffers 5
//--- The indicator has 2 plots
#property indicator_plots 2
//--- Plot 1 is a line
#property indicator_type1 DRAW_LINE
//--- The color of the line is clrDodgerRed
#property indicator_color1 clrDodgerRed
//--- Plot 2 is drawn as Japanese candlesticks
#property indicator_type2 DRAW_CANDLES
//--- The color of the candlesticks is clrDodgerBlue
#property indicator_color2 clrDodgerBlue
//--- 5 arrays for indicator buffers
double LineBuffer[];
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

The order of the plots has changed, and now the line comes first, followed by Japanese candlesticks. Therefore, the order of the buffers is appropriate - first we announce a buffer for the line with the zero index, and then four buffers for the candlesticks.

```
SetIndexBuffer(0,LineBuffer,INDICATOR_DATA); // The first buffer corresponds to in
//--- Binding arrays with indicator buffers for the candlesticks
SetIndexBuffer(1,OBuffer,INDICATOR_DATA); // The second buffer corresponds to i
SetIndexBuffer(2,HBuffer,INDICATOR_DATA); // The third buffer corresponds to in
SetIndexBuffer(3,LBuffer,INDICATOR_DATA); // The fourth buffer corresponds to i
SetIndexBuffer(4,CBuffer,INDICATOR_DATA); // The fifth buffer corresponds to in
```

The number of buffers and plots can be set only by using compiler directives, it is impossible to change these properties dynamically using functions.

Color Versions of Styles

As can be seen in the table, the styles are divided into two groups. The first group includes styles in whose name there is no word **COLOR**, we call these styles basic:

- DRAW_LINE
- DRAW_SECTION
- DRAW_HISTOGRAM
- DRAW_HISTOGRAM2
- DRAW_ARROW
- DRAW_ZIGZAG
- DRAW_FILLING
- DRAW_BARS
- DRAW_CANDLES

In the second group, the style names contain the word **COLOR**, let's call them color versions:

- DRAW_COLOR_LINE
- DRAW_COLOR_SECTION
- DRAW_COLOR_HISTOGRAM
- DRAW_COLOR_HISTOGRAM2
- DRAW_COLOR_ARROW
- DRAW_COLOR_ZIGZAG
- DRAW_COLOR_BARS
- DRAW_COLOR_CANDLES

All color versions of styles differ from the basic ones in that they allow specifying a color for each part of the plotting. The minimal part of plotting is a bar, so we can say that the color versions allow setting the color on each bar.

Exceptions are styles [DRAW_NONE](#) and [DRAW_FILLING](#), they do not have color versions.

To set the plotting color on each bar, an additional buffer for storing the color index has been added to the color version. These indices indicate the number of a color in a special array, which contains a predefined set of colors. The size of the array of colors is 64. This means that each color version of a style allows painting a plot in 64 different colors.

The set and the number of colors in the special array of colors can be set via a compiler directive `#property indicator_color`, where you can specify all the necessary colors separated by commas. For example, such an entry in an indicator:

```
//--- Define 8 colors for coloring candlesticks (they are stored in the special array)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
```

It states that for plotting 1, 8 colors are set, which will be placed in a special array. Further in the program we will not specify the color of the plotting, but only its index. If we want to set red color for the bar number **K**, the color index value from an array should be set in the color buffer of the indicator. The red color is specified first in the directive, it corresponds to the index number 0.

```
//--- set the candlestick color clrRed
col_buffer[buffer_index]=0;
```

The set of colors is not given once and for all, it can be changed dynamically using `PlotIndexSetInteger()`. Example:


```
//--- Set the color for each index as the property PLOT_LINE_COLOR
PlotIndexSetInteger(0,           // The number of a graphical style
                    PLOT_LINE_COLOR, // Property identifier
                    plot_color_ind, // The index of the color, where we v
                    color_array[i]); // A new color
```

Properties of the indicator and plotting

For indicator plots, properties can be set by means of [compiler directives](#) and using the appropriate functions. Read more information about this in [Connection between Indicator Properties and Functions](#). Dynamic change of indicator properties using special functions allows creating more flexible custom indicators.

Start of Indicator Drawing on the Chart

In many cases, according to the conditions of the algorithm, it is impossible to start calculating the indicator values immediately with the current bar, since it is necessary to provide a minimum number of previous bars available in history. For example, many types of smoothing imply using an array of prices over the previous N bars, and on the basis of these values, the indicator value on the current bar is calculated.

In such cases, either there is no way to calculate the indicator values for the first N bars, or these values are not intended to be displayed on the chart and are only subsidiary for calculating further values. To avoid plotting of the indicator on the first N bars of the history, set the N value to the [PLOT_DRAW_BEGIN](#) property for the corresponding plot:

```
//--- Binding arrays with indicator buffers for the candlesticks
PlotIndexSetInteger(number_of_plot, PLOT_DRAW_BEGIN, N);
```

Here:

- number_of_plot - a value from zero to indicator_plots-1 (numbering of plots starts with zero).
- N - the number of first bars in the history, on which the indicator should not be displayed on the chart.

DRAW_NONE

The DRAW_NONE style is designed for use in cases where it is necessary to calculate the values of a buffer and show them in the Data Window, but plotting on the chart is not required. To set up the accuracy use the expression `IndicatorSetInteger(INDICATOR_DIGITS,num_chars)` in the [OnInit\(\)](#) function:

```
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
    //--- Set the accuracy of values to be displayed in the Data Window
    IndicatorSetInteger(INDICATOR_DIGITS,0);
    //---
    return(INIT_SUCCEEDED);
}
```

The number of buffers required for plotting DRAW_NONE is 1.

An example of the indicator that shows the number of the bar on which the mouse currently hovers in the Data Window. The numbering corresponds to the timeseries, meaning the current unfinished bar has the zero index, and the oldest bar has the largest index.



Note that despite the fact that, for red color is set plotting #1, the indicator does not draw anything on the chart.

```
//+-----+
//|                                     DRAW_NONE.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
```



```
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots  1
//--- plot Invisible
#property indicator_label1  "Bar Index"
#property indicator_type1   DRAW_NONE
#property indicator_style1  STYLE_SOLID
#property indicator_color1  clrRed
#property indicator_width1  1
//--- indicator buffers
double      InvisibleBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- Binding an array and an indicator buffer
    SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- Set the accuracy of values to be displayed in the Data Window
    IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static datetime lastbar=0;
//--- If this is the first calculation of the indicator
    if(prev_calculated==0)
    {
        //--- Renumber the bars for the first time
        CalcValues(rates_total,close);
        //--- Remember the opening time of the current bar in lastbar
        lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
    }
}
```



```

    }
    else
    {
        //--- If a new bar has appeared, its open time differs from lastbar
        if(lastbar!=SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE))
        {
            //--- Renumber the bars once again
            CalcValues(rates_total,close);
            //--- Update the opening time of the current bar in lastbar
            lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}

//+-----+
//| Number the bars like in a timeseries |
//+-----+
void CalcValues(int total,double const &array[])
{
    //--- Set indexing of the indicator buffer like in a timeseries
    ArraySetAsSeries(InvisibleBuffer,true);
    //--- Fill in each bar with its number
    for(int i=0;i<total;i++) InvisibleBuffer[i]=i;
}

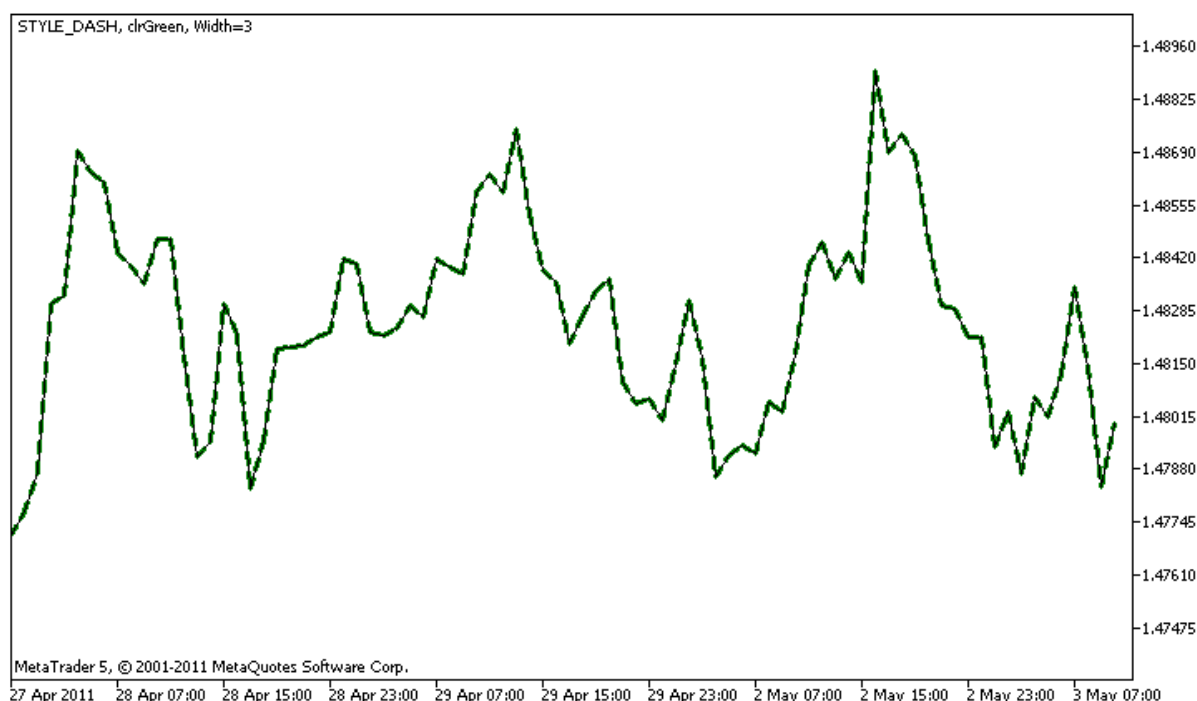
```


DRAW_LINE

DRAW_LINE draws a line of the specified color by the values of the indicator buffer. The width, style and color of the line can be set using the [compiler directives](#) and dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The number of buffers required for plotting DRAW_LINE is 1.

An example of the indicator that draws a line using Close prices of bars. The line color, width and style change randomly every N=5 ticks.



Note that initially for `plot1` with DRAW_LINE the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these three properties are set randomly. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_LINE.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_LINE"
#property description "It draws a line of a specified color at Close prices"
#property description "Color, width and style of lines is changed randomly"
#property description "after every N ticks"
```



```

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- Line properties are set using the compiler directives
#property indicator_label1 "Line" // Name of a plot for the Data Window
#property indicator_type1 DRAW_LINE // Type of plotting is line
#property indicator_color1 clrRed // Line color
#property indicator_style1 STYLE_SOLID // Line style
#property indicator_width1 1 // Line Width
//--- input parameter
input int N=5; // Number of ticks to change
//--- An indicator buffer for the plot
double LineBuffer[];
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- Binding an array and an indicator buffer
SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//--- Initializing the generator of pseudo-random numbers
MathSrand(GetTickCount());
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {

```



```

    //--- Change the line properties
    ChangeLineAppearance();
    //--- Reset the counter of ticks to zero
    ticks=0;
}

//--- Block for calculating indicator values
for(int i=0;i<rates_total;i++)
{
    LineBuffer[i]=close[i];
}

//--- Return the prev_calculated value for the next call of the function
return(rates_total);
}

//+-----+
//| Changes the appearance of the drawn line in the indicator |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the line properties
    string comm="";
    //--- A block for changing the color of the line
    //--- Get a random number
    int number=MathRand();
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
    //--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- Write the line color
    comm=comm+(string)colors[color_index];

    //--- A block for changing the width of the line
    number=MathRand();
    //--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
    //--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- Write the line width
    comm=comm+", Width="+IntegerToString(width);

    //--- A block for changing the style of the line
    number=MathRand();
    //--- The divisor is equal to the size of the styles array
    size=ArraySize(styles);
    //--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;

```



```
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
    comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}
```


DRAW_SECTION

DRAW_SECTION draws sections of the specified color by the values of the indicator buffer. The width, color and style of the line can be specified like for the [DRAW_LINE](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

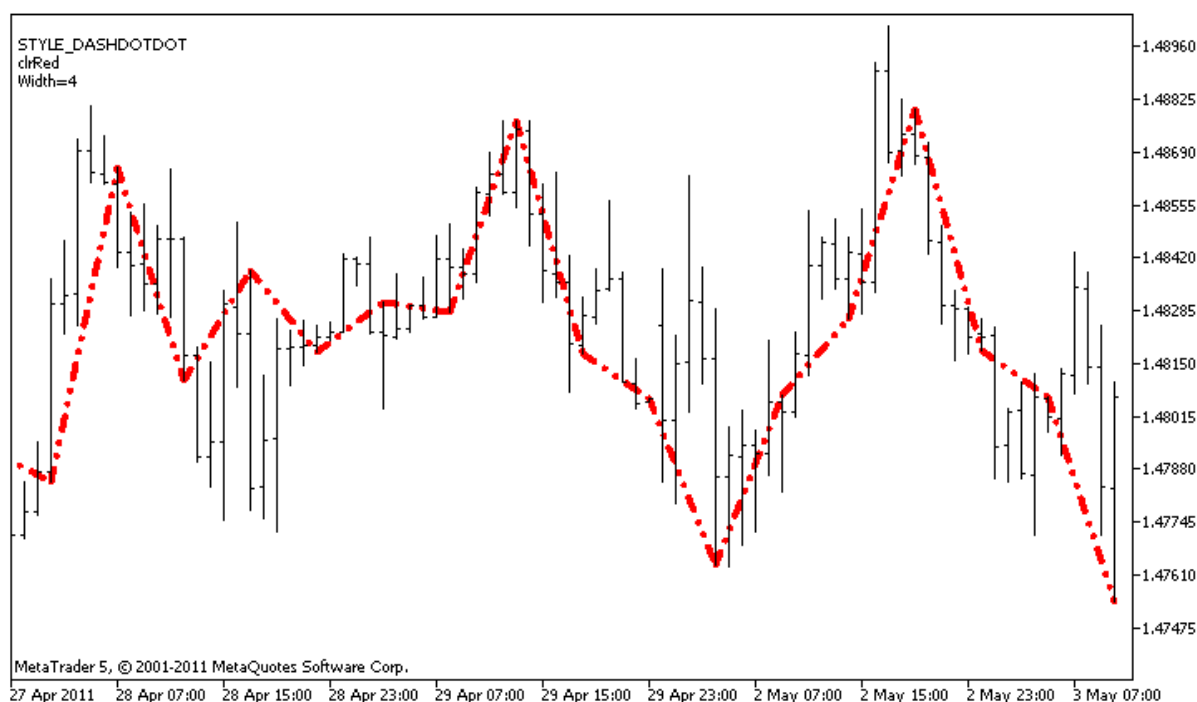
Sections are drawn from one non-empty value to another non-empty value of the indicator buffer, empty values are ignored. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property: For example, if the indicator should be drawn as a sequence of sections on non-zero values, then you need to set the zero value as an empty one:

```
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to the elements that should not be plotted.

The number of buffers required for plotting DRAW_SECTION is 1.

An example of the indicator that draws sections between the High and Low prices. The color, width and style of all sections change randomly every N ticks.



Note that initially for plot1 with DRAW_SECTION the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these three properties are set randomly. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_SECTION.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
```



```
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_SECTION"
#property description "Draws straight sections every bars bars"
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Section
#property indicator_label1 "Section"
#property indicator_type1  DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1 1
//--- input parameter
input int      bars=5;           // The length of sections in bars
input int      N=5;             // The number of ticks to change the style of section
//--- An indicator buffer for the plot
double         SectionBuffer[];
//--- An auxiliary variable to calculate ends of sections
int            divider;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- Binding an array and an indicator buffer
SetIndexBuffer(0,SectionBuffer,INDICATOR_DATA);
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- Check the indicator parameter
if(bars<=0)
{
PrintFormat("Invalid value of parameter bar=%d",bars);
return(INIT_PARAMETERS_INCORRECT);
}
else divider=2*bars;
//---+
return(INIT_SUCCEEDED);
}
```



```

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
    //--- Calculate ticks to change the style, color and width of the line
    ticks++;
    //--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

    //--- The number of the bar from which the calculation of indicator values starts
    int start=0;
    //--- If the indicator has been calculated before, then set start on the previous bar
    if(prev_calculated>0) start=prev_calculated-1;
    //--- Here are all the calculations of the indicator values
    for(int i=start;i<rates_total;i++)
    {
        //--- Get a remainder of the division of the bar number by 2*bars
        int rest=i%divider;
        //--- If the bar number is divisible by 2*bars
        if(rest==0)
        {
            //--- Set the end of the section at the High price of this bar
            SectionBuffer[i]=high[i];
        }
        //---If the remainder of the division is equal to bars,
        else
        {
            //--- Set the end of the section at the High price of this bar
            if(rest==bars) SectionBuffer[i]=low[i];
            //--- If nothing happened, ignore the bar - set 0
            else SectionBuffer[i]=0;
        }
    }
}

```



```

    }
    //--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}
//+-----+
//| Changes the appearance of sections in the indicator |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the line properties
    string comm="";
    //--- A block of line color change
    int number=MathRand(); // Get a random number
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
    //--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- Write the line color
    comm=comm+"\r\n"+(string)colors[color_index];

    //--- A block for changing the width of the line
    number=MathRand();
    //--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
    //--- Set the width
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

    //--- A block for changing the style of the line
    number=MathRand();
    //--- The divisor is equal to the size of the styles array
    size=ArraySize(styles);
    //--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;
    //--- Set the line style
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
    //--- Write the line style
    comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
    //--- Show the information on the chart using a comment
    Comment(comm);
}

```

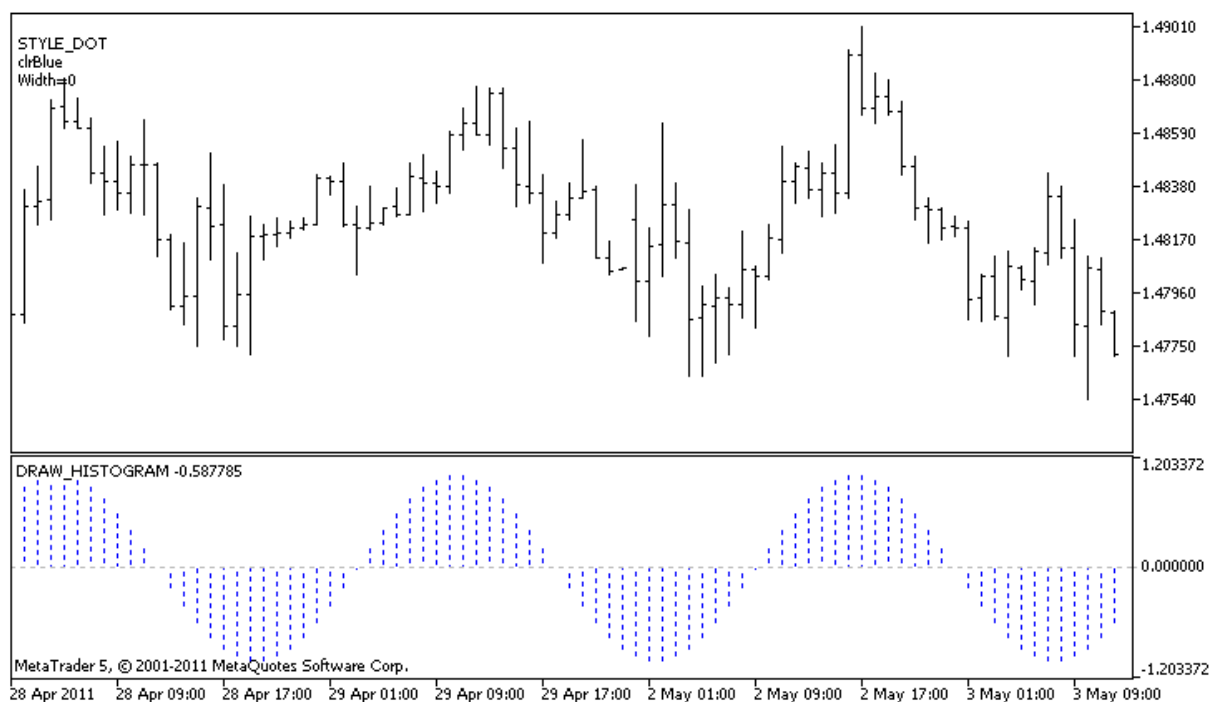

DRAW_HISTOGRAM

The DRAW_HISTOGRAM style draws a histogram as a sequence of columns of a specified color from zero to a specified value. Values are taken from the indicator buffer. The width, color and style of the column can be specified like for the [DRAW_LINE](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

Since a column from the zero level is drawn on each bar, DRAW_HISTOGRAM should better be used in a separate chart window. Most often this type of plotting is used to create indicators of the oscillator type, for example, [Bears Power](#) or [OsMA](#). For the empty non-displayable values the zero value should be specified.

The number of buffers required for plotting DRAW_HISTOGRAM is 1.

An example of the indicator that draws a sinusoid of a specified color based on the [MathSin\(\) function](#). The color, width and style of all histogram columns change randomly each N ticks. The bars parameter specifies the period of the sinusoid, that is after the specified number of bars the sinusoid will repeat the cycle.



Note that initially for plot1 with DRAW_HISTOGRAM the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these three properties are set randomly. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_HISTOGRAM.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
```



```

#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars in the cycle of the

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=30;           // The period of a sinusoid in bars
input int      N=5;              // The number of ticks to change the histogram
//--- indicator buffers
double         HistogramBuffer[];
//--- A factor to get the 2Pi angle in radians, when multiplied by the bars parameter
double         multiplier;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- Calculate the multiplier
    if(bars>1)multiplier=2.*M_PI/bars;
    else
    {
        PrintFormat("Set the value of bars=%d greater than 1",bars);
        //--- Early termination of the indicator
        return(INIT_PARAMETERS_INCORRECT);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |

```



```
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
    //--- Calculate ticks to change the style, color and width of the line
    ticks++;
    //--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

    //--- Calculate the indicator values
    int start=0;
    //--- If already calculated during the previous starts of OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculation
    //--- Fill in the indicator buffer with values
    for(int i=start;i<rates_total;i++)
    {
        HistogramBuffer[i]=sin(i*multiplier);
    }
    //--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}
//+-----+
//| Changes the appearance of lines in the indicator |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the line properties
    string comm="";
    //--- A block for changing the color of the line
    int number=MathRand(); // Get a random number
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
```



```

//--- Set the color as the PLOT_LINE_COLOR property
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
number=MathRand();
//--- Get the width of the remainder of integer division
int width=number%5; // The width is set from 0 to 4
//--- Set the width
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
number=MathRand();
//--- The divisor is equal to the size of the styles array
size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
int style_index=number%size;
//--- Set the line style
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- Show the information on the chart using a comment
Comment(comm);
}

```


DRAW_HISTOGRAM2

The DRAW_HISTOGRAM2 style draws a histogram of a specified color - vertical segments using the values of two indicator buffers. The width, color and style of the segments can be specified like for the [DRAW_LINE](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

The DRAW_HISTOGRAM2 style can be used in a separate subwindow of a chart and in its main window. For empty values nothing is drawn, all the values in the indicator buffers need to be set explicitly. Buffers are not initialized with a zero value.

The number of buffers required for plotting DRAW_HISTOGRAM2 is 2.

An example of the indicator that plots a vertical segment of the specified color and width between the Open and Close prices of each bar. The color, width and style of all histogram columns change randomly each N ticks. During the start of the indicator, in the [OnInit\(\)](#) function, the number of the day of week for which the histogram will not be drawn - invisible_day - is set randomly. For this purpose an empty value is set [PLOT_EMPTY_VALUE=0](#):

```
//--- Set an empty value
PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```



Note that initially for plot1 with DRAW_HISTOGRAM2 the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these three properties are set randomly. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_HISTOGRAM2.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
```



```
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot Histogram_2
#property indicator_label1 "Histogram_2"
#property indicator_type1  DRAW_HISTOGRAM2
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;           // The number of ticks to change the histogram
//--- indicator buffers
double        Histogram_2Buffer1[];
double        Histogram_2Buffer2[];
//--- The day of the week for which the indicator is not plotted
int invisible_day;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Histogram_2Buffer1,INDICATOR_DATA);
    SetIndexBuffer(1,Histogram_2Buffer2,INDICATOR_DATA);
//--- Set an empty value
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- Get a random number from 0 to 5
    invisible_day=MathRand()%6;
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
```



```

        const int prev_calculated,
        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])

{
    static int ticks=0;
    //--- Calculate ticks to change the style, color and width of the line
    ticks++;
    //--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

    //--- Calculate the indicator values
    int start=0;
    //--- To get the day of week by the open price of each bar
    MqlDateTime dt;
    //--- If already calculated during the previous starts of OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculation
    //--- Fill in the indicator buffer with values
    for(int i=start;i<rates_total;i++)
    {
        TimeToStruct(time[i],dt);
        if(dt.day_of_week==invisible_day)
        {
            Histogram_2Buffer1[i]=0;
            Histogram_2Buffer2[i]=0;
        }
        else
        {
            Histogram_2Buffer1[i]=open[i];
            Histogram_2Buffer2[i]=close[i];
        }
    }

    //--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}

//+-----+
//| Changes the appearance of lines in the indicator |
//+-----+

```



```

void ChangeLineAppearance()
{
//--- A string for the formation of information about the line properties
    string comm="";
//--- A block of line color change
    int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
    comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
    number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
//--- Set the line width
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
    number=MathRand();
//--- The divisor is equal to the size of the styles array
    size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;
//--- Set the line style
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
    comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- Add information about the day that is omitted in calculations
    comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}

```


DRAW_ARROW

The DRAW_ARROW style draws arrows of the specified color (symbols of the set [Wingdings](#)) based on the value of the indicator buffer. The width and color of the symbols can be specified like for the [DRAW_LINE](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of an indicator based on the current situation.

The symbol code is set using the [PLOT_ARROW](#) property.

```
//--- Define the symbol code from the Wingdings font to draw in PLOT_ARROW  
PlotIndexSetInteger(0,PLOT_ARROW,code);
```

The default value of PLOT_ARROW=159 (a circle).

Each arrow is actually a symbol that has the height and the anchor point, and can cover some important information on a chart (for example, the closing price at the bar). Therefore, we can additionally specify the vertical shift in pixels, which does not depend on the scale of the chart. The arrows will be shifted down by the specified number of pixels, although the values of the indicator will remain the same:

```
//--- Set the vertical shift of arrows in pixels  
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

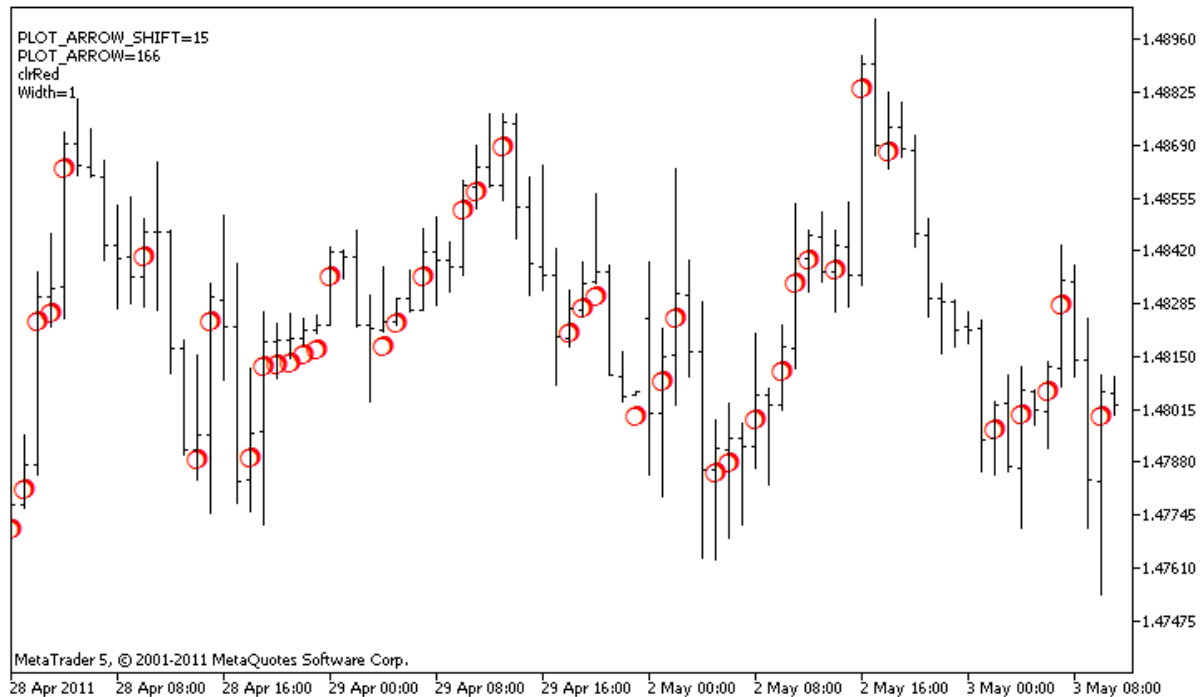
A negative value of PLOT_ARROW_SHIFT means the shift of arrows upwards, a positive values shifts the arrow down.

The DRAW_ARROW style can be used in a separate subwindow of a chart and in its main window. Empty values are not drawn and do not appear in the "Data Window", all the values in the indicator buffers should be set explicitly. Buffers are not initialized with a zero value.

```
//--- Set an empty value  
PlotIndexSetDouble(index_of_plot_DRAW_ARROW,PLOT_EMPTY_VALUE,0);
```

The number of buffers required for plotting DRAW_ARROW is 1.

An example of the indicator, which draws arrows on each bar with the close price higher than the close price of the previous bar. The color, width, shift and symbol code of all arrows are changed randomly every N ticks.



In the example, for `plot1` with the `DRAW_ARROW` style, the properties, color and size are specified using the compiler directive `#property`, and then in the `OnCalculate()` function the properties are set randomly. The `N` parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_ARROW.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ARROW"
#property description "Draws arrows set by Unicode characters, on a chart"
#property description "The color, size, shift and symbol code of the arrow are changed"
#property description "after every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Arrows
#property indicator_label1 "Arrows"
#property indicator_type1  DRAW_ARROW
#property indicator_color1  clrGreen
#property indicator_width1 1
//--- input parameters
input int      N=5;          // Number of ticks to change
```



```

input ushort   code=159;    // Symbol code to draw in DRAW_ARROW
//--- An indicator buffer for the plot
double         ArrowsBuffer[];
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ArrowsBuffer,INDICATOR_DATA);
//--- Define the symbol code for drawing in PLOT_ARROW
    PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- Set the vertical shift of arrows in pixels
    PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- Set as an empty value 0
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the color, size, shift and code of the arrow
    ticks++;
//--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- Block for calculating indicator values
    int start=1;

```



```

    if(prev_calculated>0) start=prev_calculated-1;
//--- Calculation loop
    for(int i=1;i<rates_total;i++)
    {
        //--- If the current Close price is higher than the previous one, draw an arrow
        if(close[i]>close[i-1])
            ArrowsBuffer[i]=close[i];
        //--- Otherwise specify the zero value
        else
            ArrowsBuffer[i]=0;
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Change the appearance of symbols in the indicator |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the indicator properties
    string comm="";
//--- A block for changing the arrow color
    int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
    comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the size arrows
    number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The size is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the arrow size
    comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the arrow code (PLOT_ARROW)
    number=MathRand();
//--- Get the remainder of integer division to calculate a new code of the arrow (from
    int code_add=number%20;
//--- Set the new symbol code as the result of code+code_add
    PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- Write the symbol code PLOT_ARROW
    comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

```



```
//--- A block for changing the vertical shift of arrows in pixels
number=MathRand();
//--- Get the shift as the remainder of the integer division
int shift=20-number%41;
//--- Set the new shift from -20 to 20
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- Write the shift PLOT_ARROW_SHIFT
comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- Show the information on the chart using a comment
Comment(comm);
}
```


DRAW_ZIGZAG

The DRAW_ZIGZAG style draws segments of a specified color based on the values of two indicator buffers. This style is very similar to [DRAW_SECTION](#), but unlike the latter, it allows drawing vertical segments within one bar, if values of both indicator buffers are set for this bar. The segments are plotted from a value in the first buffer to a value in the second indicator buffer. None of the buffers can contain only empty values, since in this case nothing is plotted.

The width, color and style of the line can be specified like for the [DRAW_SECTION](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

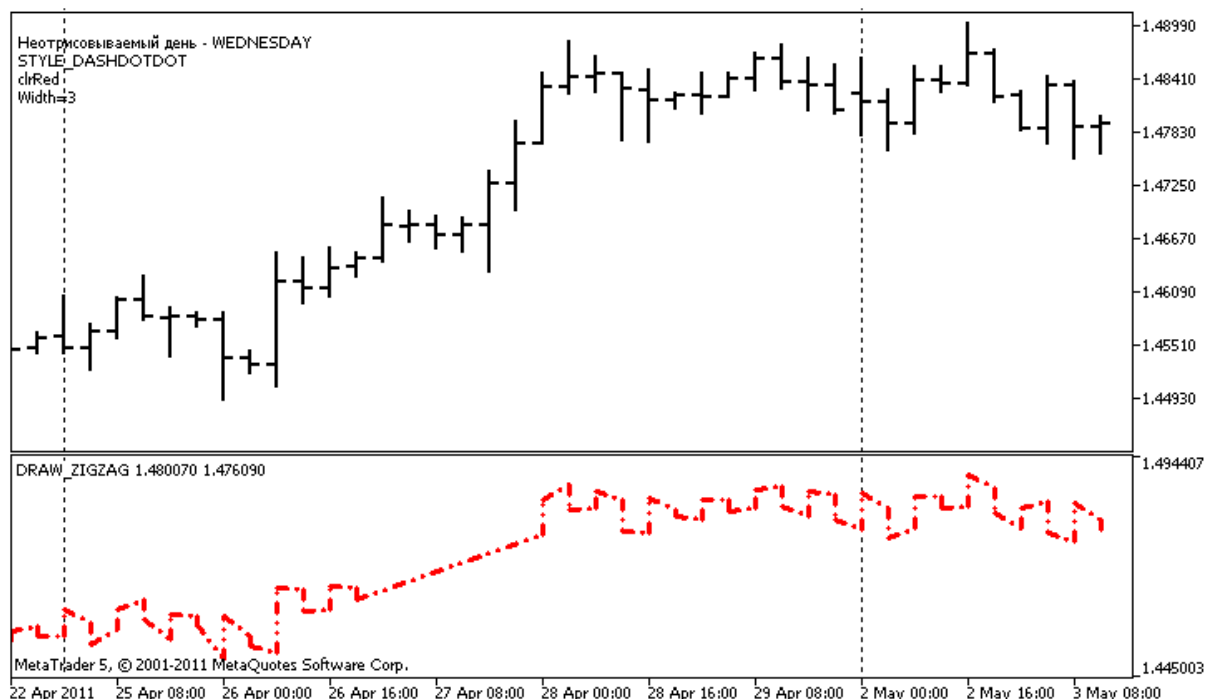
Sections are drawn from a non-empty value of one buffer to a non-empty value of another indicator buffer. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

```
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(index_of_plot_DRAW_ZIGZAG, PLOT_EMPTY_VALUE, 0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_ZIGZAG is 2.

An example of the indicator that plots a saw based on the High and Low prices. The color, width and style of the zigzag lines change randomly every N ticks.



Note that initially for `plot1` with DRAW_ZIGZAG the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these properties are set randomly. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).


```

//+-----+
//|                                     DRAW_ZIGZAG.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ZIGZAG"
#property description "It draws a \"saw\" as straight segments, skipping the bars of c
#property description "The day to skip is selected randomly during indicator start"
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ZigZag
#property indicator_label1 "ZigZag"
#property indicator_type1  DRAW_ZIGZAG
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;           // Number of ticks to change
//--- indicator buffers
double        ZigZagBuffer1[];
double        ZigZagBuffer2[];
//--- The day of the week for which the indicator is not plotted
int invisible_day;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- Binding arrays and indicator buffers
    SetIndexBuffer(0,ZigZagBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ZigZagBuffer2,INDICATOR_DATA);
//--- Get a random value from 0 to 6, for this day the indicator is not plotted
    invisible_day=MathRand()%6;
//--- The 0 (empty) value will not participate in drawing
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- The 0 (empty) value will not participate in drawing
    PlotIndexSetString(0,PLOT_LABEL,"ZigZag1;ZigZag2");

```



```

//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- The structure of time is required to get the day of week of each bar
    MqlDateTime dt;

//--- The start position of calculations
    int start=0;
//--- If the indicator was calculated on the previous tick, then start the calculation
    if(prev_calculated!=0) start=prev_calculated-1;
//--- Calculation loop
    for(int i=start;i<rates_total;i++)
    {
        //--- Write the bar open time in the structure
        TimeToStruct(time[i],dt);
        //--- If the day of the week of this bar is equal to invisible_day
        if(dt.day_of_week==invisible_day)
        {
            //--- Write empty values to buffers for this bar
            ZigZagBuffer1[i]=0;
            ZigZagBuffer2[i]=0;
        }
        //--- If the day of the week is ok, fill in the buffers
    }
}

```



```

else
{
    //--- If the bar number is even
    if(i%2==0)
    {
        //--- Write High in the 1st buffer and Low in the 2nd one
        ZigZagBuffer1[i]=high[i];
        ZigZagBuffer2[i]=low[i];
    }
    //--- The bar number is odd
    else
    {
        //--- Fill in the bar in a reverse order
        ZigZagBuffer1[i]=low[i];
        ZigZagBuffer2[i]=high[i];
    }
}

}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Changes the appearance of the zigzag segments |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the ZigZag properties
    string comm="";
    //--- A block for changing the color of the ZigZag
    int number=MathRand(); // Get a random number
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
    //--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- Write the line color
    comm=comm+"\r\n"+(string)colors[color_index];

    //--- A block for changing the width of the line
    number=MathRand();
    //--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
    //--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

    //--- A block for changing the style of the line

```



```
number=MathRand();
//--- The divisor is equal to the size of the styles array
size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- Add information about the day that is omitted in calculations
comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- Show the information on the chart using a comment
Comment(comm);
}
```


DRAW_FILLING

The DRAW_FILLING style plots a colored area between the values of two indicator buffers. In fact, this style draws two lines and fills the space between them with one of two specified colors. It is used for creating indicators that draw channels. None of the buffers can contain only empty values, since in this case nothing is plotted.

You can set two fill colors:

- the first color is used for the areas where values in the first buffer are greater than the values in the second indicator buffer;
- the second color is used for the areas where values in the second buffer are greater than the values in the first indicator buffer.

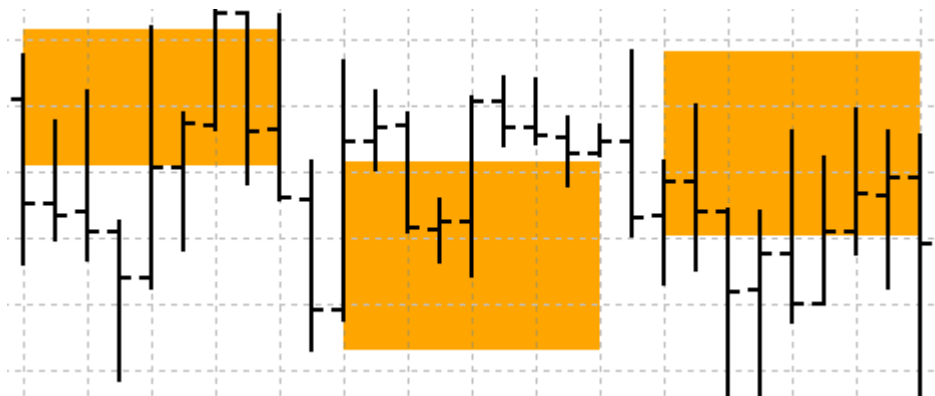
The fill color can be set using the [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is calculated for all bars, for which the values of the both indicator buffers are equal neither to 0 nor to the empty value. To specify what value should be considered as "empty", set this value in [PLOT_EMPTY_VALUE](#) property:

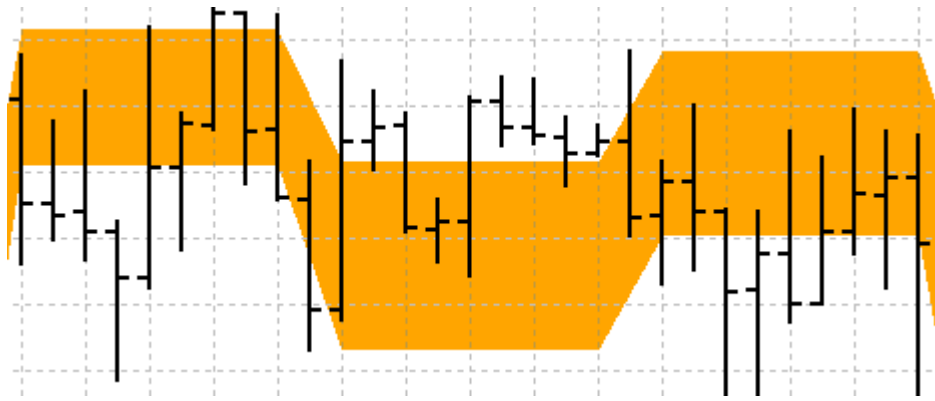
```
#define INDICATOR_EMPTY_VALUE -1.0
...
//--- INDICATOR_EMPTY_VALUE (empty value) will not participate in calculation of
PlotIndexSetDouble (DRAW_FILLING_creation_index,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_V
```

Drawing on the bars that do not participate in the indicator calculation will depend on the values in the indicator buffers:

- Bars, for which the values of both indicator buffers are equal to 0, do not participate in drawing the indicator. It means that the area with zero values is not filled out.



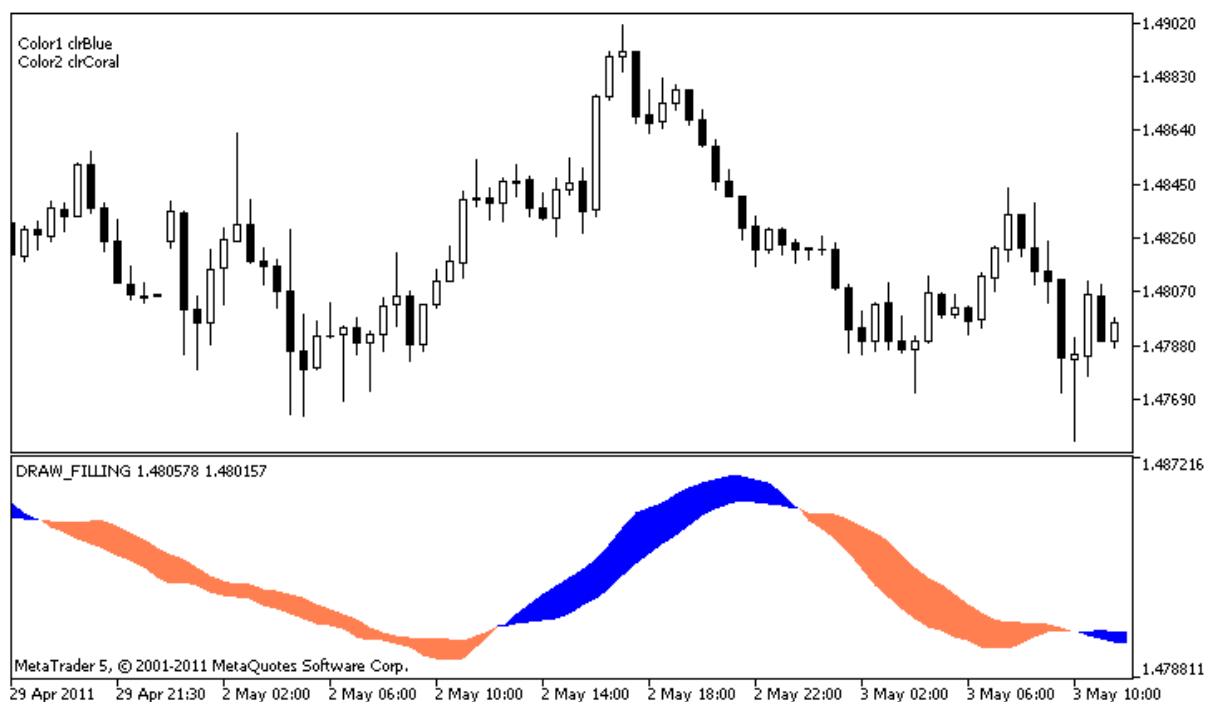
- Bars, for which the values of the indicator buffers are equal to the "empty value", participate in drawing the indicator. The area with empty values will be filled out so that to connect the areas with significant values.



It should be noted that if the "empty value" is equal to zero, the bars that do not participate in the indicator calculation are also filled out.

The number of buffers required for plotting DRAW_FILLING is 2.

An example of the indicator that draws a channel between two MAs with different averaging periods in a separate window. The change of the colors at the crossing of moving averages visually shows the change of the upward and downward trends. The colors change randomly every N ticks. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Note that initially for `plot1` with `DRAW_FILLING` the properties are set using the compiler directive `#property`, and then in the `OnCalculate()` function new colors are set randomly.

```
//+-----+
//|                                     DRAW_FILLING.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
```



```

#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_FILLING"
#property description "It draws a channel between two MAs in a separate window"
#property description "The fill color is changed randomly"
#property description "after every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot Intersection
#property indicator_label1 "Intersection"
#property indicator_type1  DRAW_FILLING
#property indicator_color1  clrRed,clrBlue
#property indicator_width1 1
//--- input parameters
input int      Fast=13;           // The period of a fast MA
input int      Slow=21;          // The period of a slow MA
input int      shift=1;          // A shift of MAs towards the future (positive)
input int      N=5;              // Number of ticks to change
//--- Indicator buffers
double         IntersectionBuffer1[];
double         IntersectionBuffer2[];
int fast_handle;
int slow_handle;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrAquamarine,clrBlanchedAlmond,clrBrown,clrC
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,IntersectionBuffer1,INDICATOR_DATA);
SetIndexBuffer(1,IntersectionBuffer2,INDICATOR_DATA);
//---
PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//---
fast_handle=iMA(_Symbol,_Period,Fast,0,MODE_SMA,PRICE_CLOSE);
slow_handle=iMA(_Symbol,_Period,Slow,0,MODE_SMA,PRICE_CLOSE);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,

```



```

        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
        //--- Calculate ticks to change the style, color and width of the line
        ticks++;
        //--- If a sufficient number of ticks has been accumulated
        if(ticks>=N)
        {
            //--- Change the line properties
            ChangeLineAppearance();
            //--- Reset the counter of ticks to zero
            ticks=0;
        }

        //--- Make the first calculation of the indicator, or data has changed and requires a
        if(prev_calculated==0)
        {
            //--- Copy all the values of the indicators to the appropriate buffers
            int copied1=CopyBuffer(fast_handle,0,0,rates_total,IntersectionBuffer1);
            int copied2=CopyBuffer(slow_handle,0,0,rates_total,IntersectionBuffer2);
        }
        else // Fill only those data that are updated
        {
            //--- Get the difference in bars between the current and previous start of OnCa
            int to_copy=rates_total-prev_calculated;
            //--- If there is no difference, we still copy one value - on the zero bar
            if(to_copy==0) to_copy=1;
            //--- copy to_copy values to the very end of indicator buffers
            int copied1=CopyBuffer(fast_handle,0,0,to_copy,IntersectionBuffer1);
            int copied2=CopyBuffer(slow_handle,0,0,to_copy,IntersectionBuffer2);
        }
        //--- return value of prev_calculated for next call
        return(rates_total);
    }

    //+-----+
    //| Changes the colors of the channel filling |
    //+-----+
    void ChangeLineAppearance()
    {
        //--- A string for the formation of information about the line properties
        string comm="";
        //--- A block for changing the color of the line

```



```
int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
int size=ArraySize(colors);

//--- Get the index to select a new color as the remainder of integer division
int color_index1=number%size;
//--- Set the first color as the PLOT_LINE_COLOR property
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,colors[color_index1]);
//--- Write the first color
comm=comm+"\r\nColor1 "+(string)colors[color_index1];

//--- Get the index to select a new color as the remainder of integer division
number=MathRand(); // Get a random number
int color_index2=number%size;
//--- Set the second color as the PLOT_LINE_COLOR property
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,colors[color_index2]);
//--- Write the second color
comm=comm+"\r\nColor2 "+(string)colors[color_index2];
//--- Show the information on the chart using a comment
Comment(comm);
}
```


DRAW_BARS

The DRAW_BARS style draws bars on the values of four indicator buffers, which contain the Open, High, Low and Close prices. It is used for creating custom indicators as bars, including those in a separate subwindow of a chart and on other financial instruments.

The color of bars can be set using the [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of **all** four indicator buffers are set. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

```
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(index_of_plot_DRAW_BARS,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_BARS is 4. All buffers for the plotting should go one after the other in the given order: Open, High, Low and Close. None of the buffers can contain only empty values, since in this case nothing is plotted.

An example of the indicator that draws bars on a selected financial instrument in a separate window. The color of bars changes randomly every N ticks. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for `plot1` with the DRAW_BARS style, the color is set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function the color is set randomly from an earlier prepared list.


```

//+-----+
//|                                     DRAW_BARS.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_BARS"
#property description "It draws bars of a selected symbol in a separate window"
#property description "The color and width of bars, as well as the symbol are changed"
#property description "every N ticks"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 1
//--- plot Bars
#property indicator_label1 "Bars"
#property indicator_type1  DRAW_BARS
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;           // The number of ticks to change the type
input int      bars=500;      // The number of bars to show
input bool     messages=false; // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double        BarsBuffer1[];
double        BarsBuffer2[];
double        BarsBuffer3[];
double        BarsBuffer4[];
//--- Symbol name
string symbol;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- If bars is very small - complete the work ahead of time
    if(bars<50)
    {
        Comment("Please specify a larger number of bars! The operation of the indicator
        return(INIT_PARAMETERS_INCORRECT);
    }
//--- indicator buffers mapping
    SetIndexBuffer(0,BarsBuffer1,INDICATOR_DATA);

```



```

SetIndexBuffer(1,BarsBuffer2,INDICATOR_DATA);
SetIndexBuffer(2,BarsBuffer3,INDICATOR_DATA);
SetIndexBuffer(3,BarsBuffer4,INDICATOR_DATA);
//--- The name of the symbol, for which the bars are drawn
symbol=_Symbol;
//--- Set the display of the symbol
PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;" +symbol+" High;" +symbol+" Low;" +symbol+" Close;");
IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS (" +symbol+" )");
//--- An empty value
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Select a new symbol from the Market watch window
        symbol=GetRandomSymbolName();
        //--- Change the line properties
        ChangeLineAppearance();

        int tries=0;
        //--- Make 5 attempts to fill in the buffers with the prices from symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total) && tries<5)
        {
            //--- A counter of calls of the CopyFromSymbolToBuffers() function
            tries++;
        }
        //--- Reset the counter of ticks to zero
        ticks=0;
    }
//--- return value of prev_calculated for next call

```



```

    return(rates_total);
}
//+-----+
//| Fill in the indicator buffers with prices |
//+-----+
bool CopyFromSymbolToBuffers(string name,int total)
{
    //--- In the rates[] array, we will copy Open, High, Low and Close
    MqlRates rates[];
    //--- The counter of attempts
    int attempts=0;
    //--- How much has been copied
    int copied=0;
    //--- Make 25 attempts to get a timeseries on the desired symbol
    while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates)<0)
    {
        Sleep(100);
        attempts++;
        if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
    }
    //--- If failed to copy a sufficient number of bars
    if(copied!=bars)
    {
        //--- Form a message string
        string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                                name,
                                copied,
                                bars
                                );

        //--- Show a message in a comment in the main chart window
        Comment(comm);
        //--- Show the message
        if(messages) Print(comm);
        return(false);
    }
    else
    {
        //--- Set the display of the symbol
        PlotIndexSetString(0,PLOT_LABEL,name+" Open;" +name+" High;" +name+" Low;" +name+"
        IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS (" +name+" )");
    }
    //--- Initialize buffers with empty values
    ArrayInitialize(BarsBuffer1,0.0);
    ArrayInitialize(BarsBuffer2,0.0);
    ArrayInitialize(BarsBuffer3,0.0);
    ArrayInitialize(BarsBuffer4,0.0);
    //--- Copy prices to the buffers
    for(int i=0;i<copied;i++)
    {

```



```

    //--- Calculate the appropriate index for the buffers
    int buffer_index=total-copied+i;
    //--- Write the prices to the buffers
    BarsBuffer1[buffer_index]=rates[i].open;
    BarsBuffer2[buffer_index]=rates[i].high;
    BarsBuffer3[buffer_index]=rates[i].low;
    BarsBuffer4[buffer_index]=rates[i].close;
}
return(true);
}

//+-----+
//| Randomly returns a symbol from the Market Watch |
//+-----+
string GetRandomSymbolName()
{
    //--- The number of symbols shown in the Market watch window
    int symbols=SymbolsTotal(true);
    //--- The position of a symbol in the list - a random number from 0 to symbols
    int number=MathRand()%symbols;
    //--- Return the name of a symbol at the specified position
    return SymbolName(number,true);
}

//+-----+
//| Changes the appearance of bars |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the bar properties
    string comm="";
    //--- A block for changing the color of bars
    int number=MathRand(); // Get a random number
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
    //--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- Write the line color
    comm=comm+"\r\n"+(string)colors[color_index];

    //--- A block for changing the width of bars
    number=MathRand();
    //--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
    //--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

```



```
//--- Write the symbol name
comm="\r\n"+symbol+comm;

//--- Show the information on the chart using a comment
Comment(comm);
}
```


DRAW_CANDLES

The DRAW_CANDLES style draws candlesticks on the values of four indicator buffers, which contain the Open, High, Low and Close prices. It is used for creating custom indicators as a sequence of candlesticks, including those in a separate subwindow of a chart and on other financial instruments.

The color of candlesticks can be set using the [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of all four indicator buffers are set. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

```
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(index_of_plot_DRAW_CANDLES,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_CANDLES is 4. All buffers for the plotting should go one after the other in the given order: Open, High, Low and Close. None of the buffers can contain only empty values, since in this case nothing is plotted.

You can set up to three colors for the DRAW_CANDLES style affecting the candle look. If only one color is set, it is applied to all candles on a chart.

```
//--- identical candles with a single color applied to them
#property indicator_label1 "One color candles"
#property indicator_type1 DRAW_CANDLES
//--- only one color is specified, therefore all candles are of the same color
#property indicator_color1 clrGreen
```

If two comma-separated colors are specified, the first one is applied to candle outlines, while the second one is applied to the body.

```
//--- different colors for candles and wicks
#property indicator_label1 "Two color candles"
#property indicator_type1 DRAW_CANDLES
//--- green is applied to wicks and outlines, while white is applied to the body
#property indicator_color1 clrGreen,clrWhite
```

Specify three comma-separated colors so that rising and falling candles are displayed differently. In that case, the first color is applied to the candle outlines, while the second and third ones - to bullish and bearish candles.

```
//--- different colors for candles and wicks
#property indicator_label1 "One color candles"
#property indicator_type1 DRAW_CANDLES
//--- wicks and outlines are green, bullish candle body is white, while bearish candle
#property indicator_color1 clrGreen,clrWhite,clrRed
```

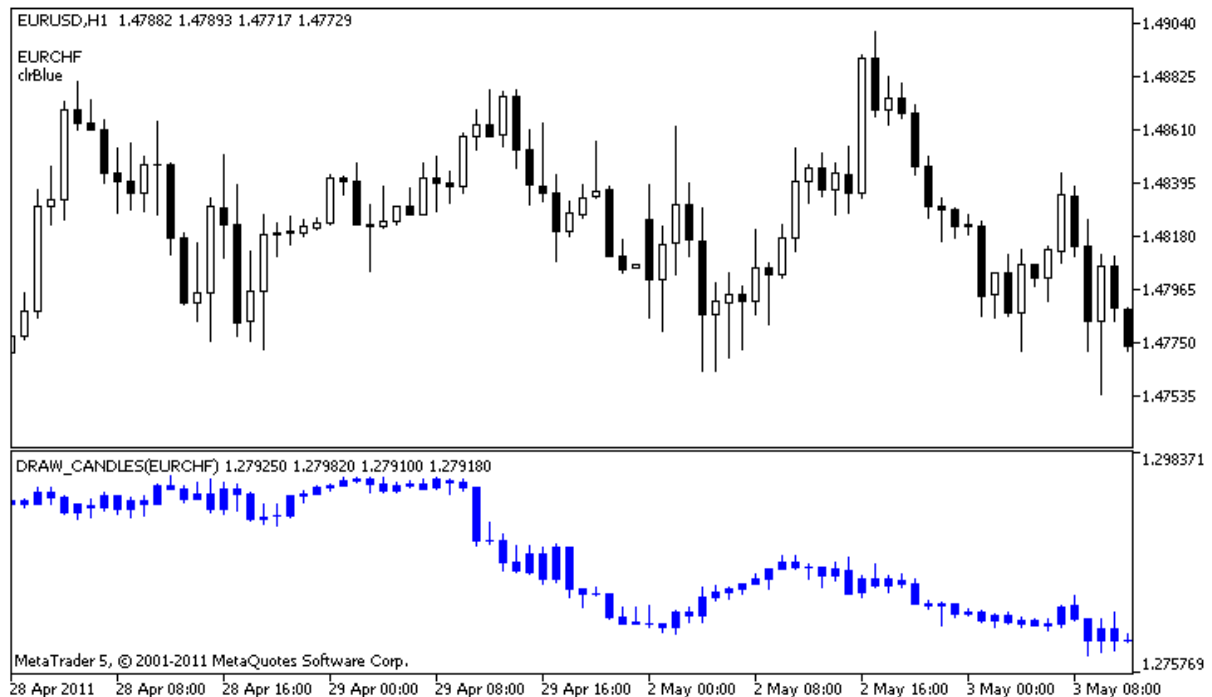
Thus, the DRAW_CANDLES style allows you to create custom candle coloring options. Besides, all colors can be changed dynamically during the indicator operation using the PlotIndexSetInteger function

(composition_index_DRAW_CANDLES, PLOT_LINE_COLOR, modifier_index, color), where modifier_index may have the following values:

- 0 - colors of outlines and wicks
- 1- bullish candle body color
- 2 - bearish candle body color

```
//--- set the color of outlines and wicks
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrBlue);
//--- set the bullish body color
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrGreen);
//--- set the bearish body color
PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrRed);
```

An example of the indicator that draws candlesticks for a selected financial instrument in a separate window. The color of candlesticks changes randomly every N ticks. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for plot1, the color is set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function the color is set randomly from an earlier prepared list.

```
//+-----+
//|                                     DRAW_CANDLES.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```



```

#property description "An indicator to demonstrate DRAW_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate window"
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are"
#property description "randomly every N ticks"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 1
//--- plot Bars
#property indicator_label1 "DRAW_CANDLES1"
#property indicator_type1 DRAW_CANDLES
#property indicator_color1 clrGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- input parameters
input int N=5; // The number of ticks to change the type
input int bars=500; // The number of bars to show
input bool messages=false; // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double Candle1Buffer1[];
double Candle1Buffer2[];
double Candle1Buffer3[];
double Candle1Buffer4[];
//--- Symbol name
string symbol;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- If bars is very small - complete the work ahead of time
if(bars<50)
{
Comment("Please specify a larger number of bars! The operation of the indicator
return(INIT_PARAMETERS_INCORRECT);
}
//--- indicator buffers mapping
SetIndexBuffer(0,Candle1Buffer1,INDICATOR_DATA);
SetIndexBuffer(1,Candle1Buffer2,INDICATOR_DATA);
SetIndexBuffer(2,Candle1Buffer3,INDICATOR_DATA);
SetIndexBuffer(3,Candle1Buffer4,INDICATOR_DATA);
//--- An empty value
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- The name of the symbol, for which the bars are drawn
symbol=_Symbol;

```



```

//--- Set the display of the symbol
    PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symbol+" Close;");
    IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_CANDLES("+symbol+")");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=INT_MAX-100;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Select a new symbol from the Market watch window
        symbol=GetRandomSymbolName();
        //--- Change the form
        ChangeLineAppearance();
        //--- Select a new symbol from the Market watch window
        int tries=0;
        //--- Make 5 attempts to fill in the buffers of plot1 with the prices from symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            Candle1Buffer1,Candle1Buffer2,Candle1Buffer3,Candle1Buffer4)
            && tries<5)
        {
            //--- A counter of calls of the CopyFromSymbolToBuffers() function
            tries++;
        }
        //--- Reset the counter of ticks to zero
        ticks=0;
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Fills in the specified candlestick |
//+-----+

```



```

bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[]
                             )
{
    //--- In the rates[] array, we will copy Open, High, Low and Close
    MqlRates rates[];
    //--- The counter of attempts
    int attempts=0;
    //--- How much has been copied
    int copied=0;
    //--- Make 25 attempts to get a timeseries on the desired symbol
    while(attempts<25 && (copied=CopyRates(name, _Period, 0, bars, rates))<0)
    {
        Sleep(100);
        attempts++;
        if(messages) PrintFormat("%s CopyRates(%s) attempts=%d", __FUNCTION__, name, attempts);
    }
    //--- If failed to copy a sufficient number of bars
    if(copied!=bars)
    {
        //--- Form a message string
        string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                                name,
                                copied,
                                bars
                                );

        //--- Show a message in a comment in the main chart window
        Comment(comm);
        //--- Show the message
        if(messages) Print(comm);
        return(false);
    }
    else
    {
        //--- Set the display of the symbol
        PlotIndexSetString(plot_index, PLOT_LABEL, name+" Open;" + name+" High;" + name+" Low;
    }
    //--- Initialize buffers with empty values
    ArrayInitialize(buff1, 0.0);
    ArrayInitialize(buff2, 0.0);
    ArrayInitialize(buff3, 0.0);
    ArrayInitialize(buff4, 0.0);
    //--- On each tick copy prices to buffers
    for(int i=0; i<copied; i++)

```



```

    {
        //--- Calculate the appropriate index for the buffers
        int buffer_index=total-copied+i;
        //--- Write the prices to the buffers
        buff1[buffer_index]=rates[i].open;
        buff2[buffer_index]=rates[i].high;
        buff3[buffer_index]=rates[i].low;
        buff4[buffer_index]=rates[i].close;
    }
    return(true);
}

//+-----+
//| Randomly returns a symbol from the Market Watch |
//+-----+
string GetRandomSymbolName()
{
    //--- The number of symbols shown in the Market watch window
    int symbols=SymbolsTotal(true);
    //--- The position of a symbol in the list - a random number from 0 to symbols
    int number=MathRand()%symbols;
    //--- Return the name of a symbol at the specified position
    return SymbolName(number,true);
}

//+-----+
//| Changes the appearance of bars |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the bar properties
    string comm="";
    //--- A block for changing the color of bars
    int number=MathRand(); // Get a random number
    //--- The divisor is equal to the size of the colors[] array
    int size=ArraySize(colors);
    //--- Get the index to select a new color as the remainder of integer division
    int color_index=number%size;
    //--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- Write the color
    comm=comm+"\r\n"+(string)colors[color_index];
    //--- Write the symbol name
    comm="\r\n"+symbol+comm;
    //--- Show the information on the chart using a comment
    Comment(comm);
}

```


DRAW_COLOR_LINE

The DRAW_COLOR_LINE value is a colored variant of the [DRAW_LINE](#) style; it also draws a line using the values of the indicator buffer. But this style, like all color styles with the word **COLOR** in their title has an additional special indicator buffer that stores the color index (number) from a specially set array of colors. Thus, the color of each line segment can be defined by specifying the color index of the index to draw the line at this bar.

The width, style and colors of lines can be set using the [compiler directives](#) and dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

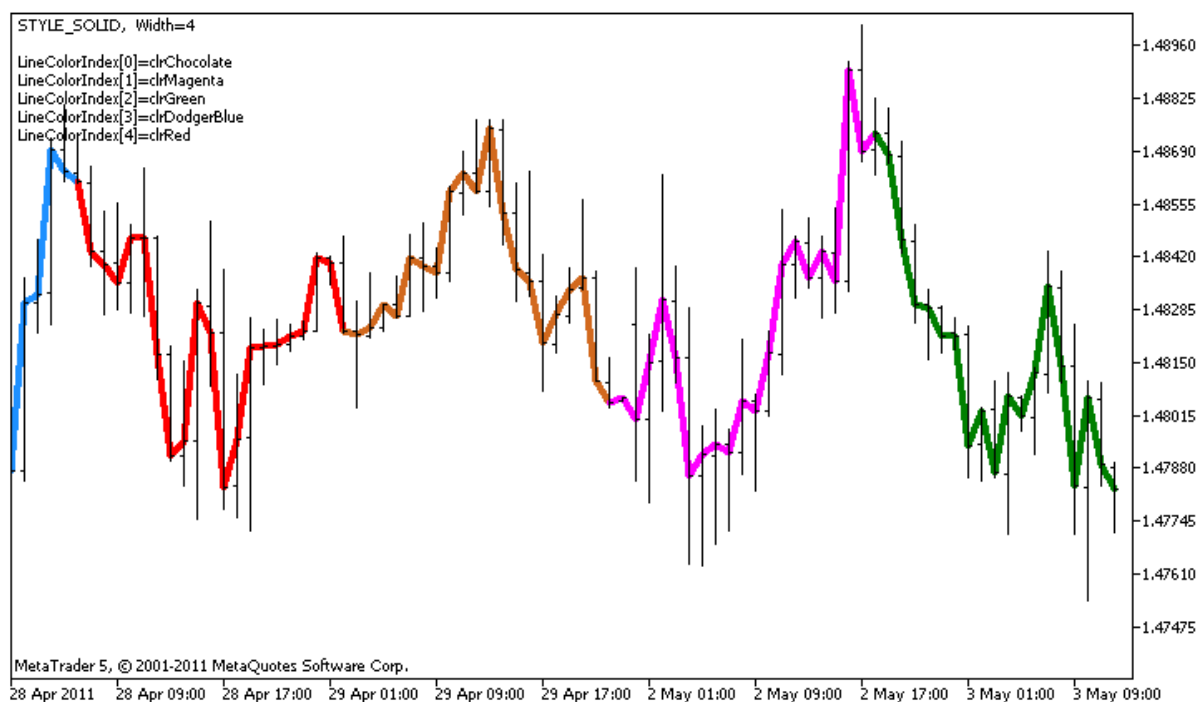
The number of buffers required for plotting DRAW_COLOR_LINE is 2.

- one buffer to store the indicator values used for drawing a line;
- one buffer to store the index of the color of the line on each bar.

Colors can be specified by the compiler directive `#property indicator_color1` separated by a comma. The number of colors cannot exceed 64.

```
//--- Define 5 colors for coloring each bar (they are stored in the special array)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 64)
```

An example of the indicator that draws a line using Close prices of bars. The line width and style change randomly every N=5 ticks.



The colors of the line segments also change randomly in the custom function ChangeColors().

```
//+-----+
//| Changes the color of line segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
```



```

{
//--- The number of colors
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0,                // The number of a graphical style
                           PLOT_LINE_COLOR,  // Property identifier
                           plot_color_ind,    // The index of the color, where we want to set
                           cols[i]);          // A new color

        //--- Write the colors
        comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(cols[i]));
        ChartSetString(0,CHART_COMMENT,comm);
    }
//---
}

```

The example shows the feature of the "color" versions of indicators - to change the color of a line segment, you do not need to change values in the ColorLineColors[] buffer (which contains the color indexes). All you need to do is set new colors in a special array. This allows you to quickly change the color once for the entire plotting, changing only a small array of colors using the [PlotIndexSetInteger\(\)](#) function.

Note that initially for plot1 with DRAW_COLOR_LINE the properties are set using the compiler directive [#property](#), and then in the [OnCalculate\(\)](#) function these three properties are set randomly.

The N and Length (the length of color segments in bars) parameters are set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```

//+-----+
//|                                     DRAW_COLOR_LINE.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_LINE"
#property description "It draws a line on Close price in colored pieces of 20 bars each"
#property description "The width, style and color of the line parts are changed randomly"

```



```

#property description "every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ColorLine
#property indicator_label1 "ColorLine"
#property indicator_type1 DRAW_COLOR_LINE
//--- Define 5 colors for coloring each bar (they are stored in the special array)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 64)
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int N=5; //Number of ticks to change
input int Length=20; // The length of each color part in bars
int line_colors=5; // The number of set colors is 5 - see #property indicator_color1
//--- A buffer for plotting
double ColorLineBuffer[];
//--- A buffer for storing the line color on each bar
double ColorLineColors[];

//--- The array for storing colors contains 7 elements
color colors[]={clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGold}
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- Binding an array and an indicator buffer
SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
SetIndexBuffer(1,ColorLineColors,INDICATOR_COLOR_INDEX);
//--- Initializing the generator of pseudo-random numbers
MathSrand(GetTickCount());
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],

```



```

        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
        ///--- Calculate ticks to change the style, color and width of the line
        ticks++;
        ///--- If a critical number of ticks has been accumulated
        if(ticks>=N)
        {
            ///--- Change the line properties
            ChangeLineAppearance();
            ///--- Change the colors of line sections
            ChangeColors(colors,5);
            ///--- Reset the counter of ticks to zero
            ticks=0;
        }

        ///--- Block for calculating indicator values
        for(int i=0;i<rates_total;i++)
        {
            ///--- Write the indicator value into the buffer
            ColorLineBuffer[i]=close[i];
            ///--- Now, randomly set a color index for this bar
            int color_index=i%(5*Length);
            color_index=color_index/Length;
            ///--- For this bar, the line will have the color with the index color_index
            ColorLineColors[i]=color_index;
        }

        ///--- Return the prev_calculated value for the next call of the function
        return(rates_total);
    }
    ///+-----+
    ///| Changes the color of line segments |
    ///+-----+
    void ChangeColors(color &cols[],int plot_colors)
    {
        ///--- The number of colors
        int size=ArraySize(cols);
        ///---
        string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

        ///--- For each color index define a new color randomly
        for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
        {
            ///--- Get a random value
            int number=MathRand();
            ///--- Get an index in the col[] array as a remainder of the integer division
            int i=number%size;

```



```

//--- Set the color for each index as the property PLOT_LINE_COLOR
PlotIndexSetInteger(0, // The number of a graphical style
    PLOT_LINE_COLOR, // Property identifier
    plot_color_ind, // The index of the color, where we v
    cols[i]); // A new color

//--- Write the colors
comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
ChartSetString(0,CHART_COMMENT,comm);
}

//---
}
//+-----+
//| Changes the appearance of a displayed line in the indicator |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the line properties
    string comm="";
//--- A block for changing the width of the line
    int number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
    comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
    number=MathRand();
//--- The divisor is equal to the size of the styles array
    int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
    comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}

```


DRAW_COLOR_SECTION

The DRAW_COLOR_SECTION style is a color version of [DRAW_SECTION](#), but unlike the latter, it allows drawing sections of different colors. The DRAW_COLOR_SECTION style, like all color styles with the word **COLOR** in their title, contains an additional special indicator buffer that stores the color index (number) from a specially set array of colors. Thus, the color of each section can be defined by specifying the color index of the index of the bar that corresponds to the section end.

The width, color and style of the sections can be specified like for the [DRAW_SECTION](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

Sections are drawn from one non-empty value to another non-empty value of the indicator buffer, empty values are ignored. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property: For example, if the indicator should be drawn as a sequence of sections on non-zero values, then you need to set the zero value as an empty one:

```
//--- The 0 (empty) value will not participate in drawing  
PlotIndexSetDouble(index_of_plot_DRAW_COLOR_SECTION, PLOT_EMPTY_VALUE, 0);
```

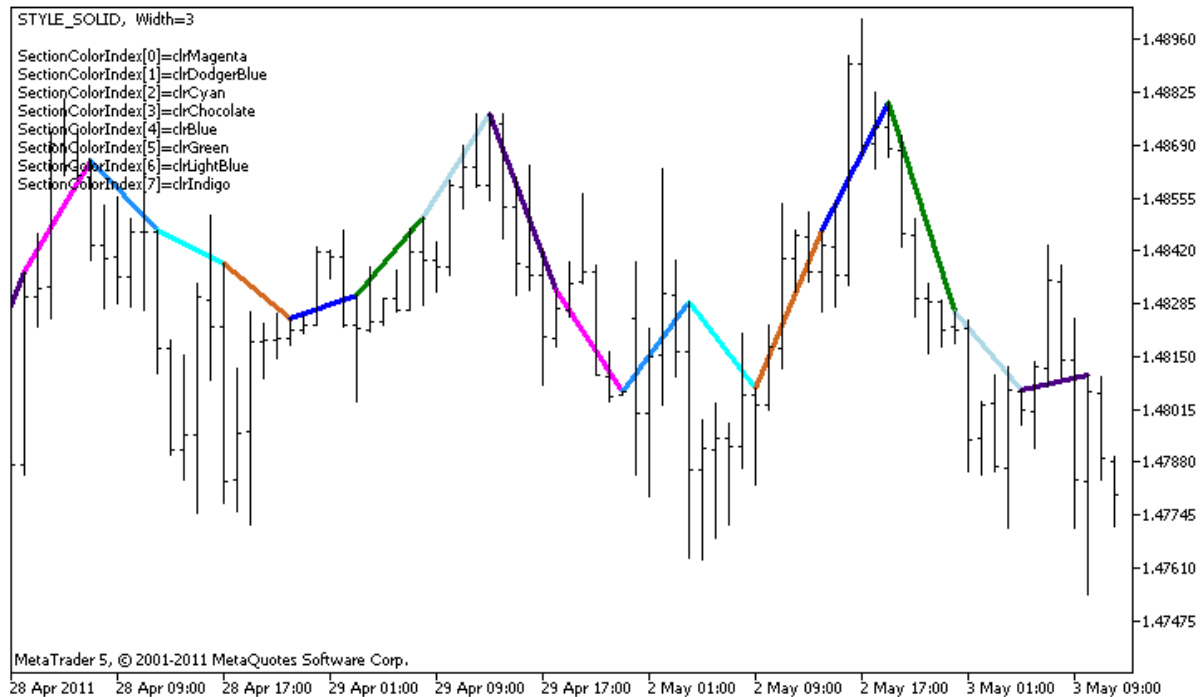
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to the elements that should not be plotted.

The number of buffers required for plotting DRAW_COLOR_SECTION is 2.

- one buffer to store the indicator values used for drawing a line;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

Colors can be specified by the compiler directive [#property indicator_color1](#) separated by a comma. The number of colors cannot exceed 64.

An example of the indicator that draws colored sections each 5 bars long, using the High price values. The color, width and style of sections change randomly every N ticks.



Note that initially for `plot1` with `DRAW_COLOR_SECTION` 8 colors are set using the compiler directive `#property`. Then in the `OnCalculate()` function, colors are set randomly from the array of colors `colors[]`.

The `N` parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_COLOR_SECTION.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_SECTION"
#property description "It draws colored sections with the length equal to the specific"
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ColorSection
#property indicator_label1 "ColorSection"
#property indicator_type1  DRAW_COLOR_SECTION
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1 clrRed,clrGold,clrMediumBlue,clrLime,clrMagenta,clrBrown,clrBlack,clrWhite
#property indicator_style1 STYLE_SOLID
```



```

#property indicator_width1 1
//--- input parameters
input int      N=5;                // Number of ticks to change
input int      bars_in_section=5;  // The length of sections in bars
//--- An auxiliary variable to calculate ends of sections
int            divider;
int            color_sections;
//--- A buffer for plotting
double         ColorSectionBuffer[];
//--- A buffer for storing the line color on each bar
double         ColorSectionColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,ColorSectionBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorSectionColors,INDICATOR_COLOR_INDEX);
    //--- The 0 (empty) value will not participate in drawing
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
    //---- The number of colors to color the sections
    int color_sections=8;    // see A comment to #property indicator_color1
    //--- Check the indicator parameter
    if(bars_in_section<=0)
    {
        PrintFormat("Invalid section length=%d",bars_in_section);
        return(INIT_PARAMETERS_INCORRECT);
    }
    else divider=color_sections*bars_in_section;
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],

```



```

        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])

{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Change colors used to plot the sections
        ChangeColors(colors,color_sections);
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- The number of the bar from which the calculation of indicator values starts
    int start=0;
//--- If the indicator has been calculated before, then set start on the previous bar
    if(prev_calculated>0) start=prev_calculated-1;
//--- Here are all the calculations of the indicator values
    for(int i=start;i<rates_total;i++)
    {
        //--- If the bar number is divisible by the section_length, it means this is the
        if(i%bars_in_section==0)
        {
            //--- Set the end of the section at the High price of this bar
            ColorSectionBuffer[i]=high[i];
            //--- A remainder of the division of the bar number by section_length*number_
            int rest=i%divider;
            //Get the number of the color = from 0 to number_of_colors-1
            int color_indext=rest/bars_in_section;
            ColorSectionColors[i]=color_indext;
        }
        //---If the remainder of the division is equal to bars,
        else
        {
            //--- If nothing happened, ignore the bar - set 0
            else ColorSectionBuffer[i]=0;
        }
    }
//--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}
//+-----+

```



```

//| Changes the color of line segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- The number of colors
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                            PLOT_LINE_COLOR, // Property identifier
                            plot_color_ind, // The index of the color, where we v
                            cols[i]); // A new color

        //--- Write the colors
        comm=comm+StringFormat("SectionColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
        ChartSetString(0,CHART_COMMENT,comm);
    }
//---
}
//+-----+
//| Changes the appearance of a displayed line in the indicator |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the line properties
    string comm="";
//--- A block for changing the width of the line
    int number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
    comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
    number=MathRand();
//--- The divisor is equal to the size of the styles array
    int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;

```



```
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
    comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}
```


DRAW_COLOR_HISTOGRAM

The DRAW_COLOR_HISTOGRAM style draws a histogram as a sequence of colored columns from zero to a specified value. Values are taken from the indicator buffer. Each column can have its own color from a predefined set of colors.

The width, color and style of the histogram can be specified like for the [DRAW_HISTOGRAM](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

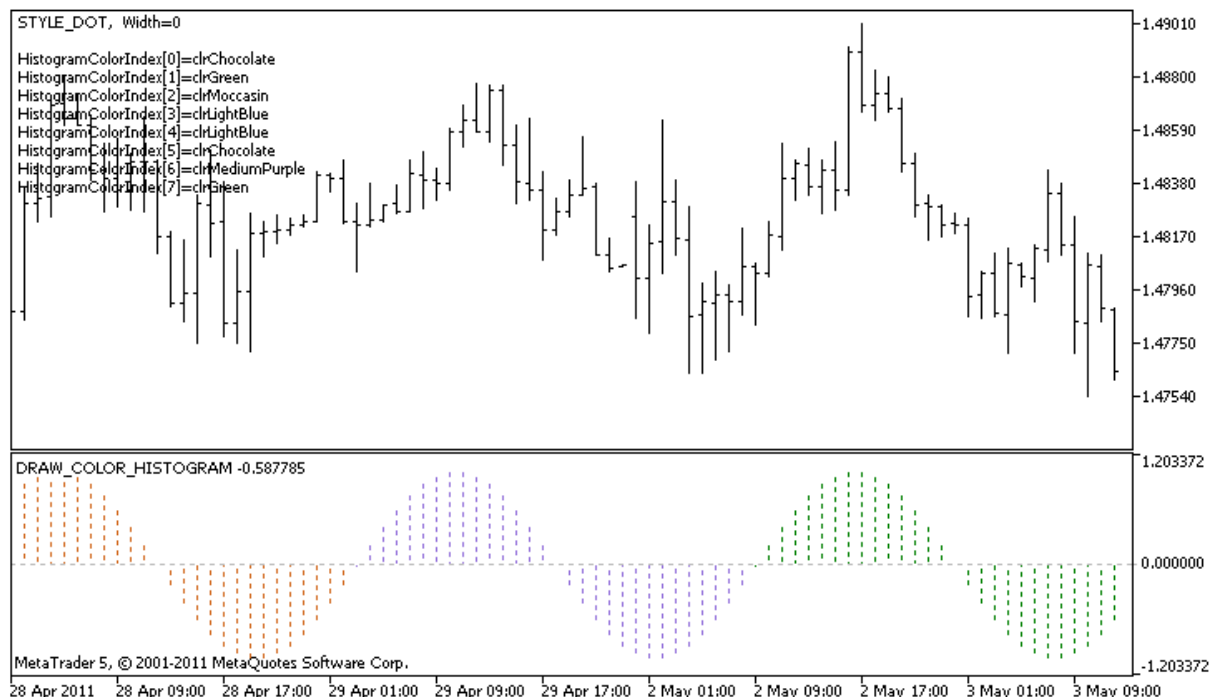
Since a column from the zero level is drawn on each bar, DRAW_COLOR_HISTOGRAM should better be used in a separate chart window. Most often this type of plotting is used to create indicators of the oscillator type, for example, [Awesome Oscillator](#) or [Market Facilitation Index](#). For the empty non-displayable values the zero value should be specified.

The number of buffers required for plotting DRAW_COLOR_HISTOGRAM is 2.

- one buffer for storing a non-zero value of the vertical segment on each bar, the second end of the segment is always on the zero line of the indicator;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

Colors can be specified using the compiler directive `#property indicator_color1` separated by a comma. The number of colors cannot exceed 64.

An example of the indicator that draws a sinusoid of a specified color based on the [MathSin\(\)](#) function. The color, width and style of all histogram columns change randomly each N ticks. The bars parameter specifies the period of the sinusoid, that is after the specified number of bars the sinusoid will repeat the cycle.



Please note that for `plot1` with the DRAW_COLOR_HISTOGRAM style, 5 colors are set using the compiler directive `#property`, and then in the [OnCalculate\(\)](#) function the colors are selected randomly

from the 14 colors stored in the colors[] array. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_COLOR_HISTOGRAM.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars to repeat the sinusoid"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- input parameters
input int      bars=30;           // The period of a sinusoid in bars
input int      N=5;              // The number of ticks to change the histogram
//--- plot Color_Histogram
#property indicator_label1  "Color_Histogram"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1  clrRed,clrGreen,clrBlue,clrYellow,clrMagenta,clrCyan,clrMagenta
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- A buffer of values
double        Color_HistogramBuffer[];
//--- A buffer of color indexes
double        Color_HistogramColors[];
//--- A factor to get the 2Pi angle in radians, when multiplied by the bars parameter
double        multiplier;
int           color_sections;
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurple
};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT}
//+-----+
//| Custom indicator initialization function |
```



```

//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Color_HistogramBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,Color_HistogramColors,INDICATOR_COLOR_INDEX);
//---- The number of colors to color the sinusoid
    color_sections=8;    // see A comment to #property indicator_color1
//--- Calculate the multiplier
    if(bars>1)multiplier=2.*M_PI/bars;
    else
    {
        PrintFormat("Set the value of bars=%d greater than 1",bars);
        //--- Early termination of the indicator
        return(INIT_PARAMETERS_INCORRECT);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Change colors used for the histogram
        ChangeColors(colors,color_sections);
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- Calculate the indicator values
    int start=0;

```



```

//--- If already calculated during the previous starts of OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculation
//--- Fill in the indicator buffer with values
    for(int i=start;i<rates_total;i++)
    {
        //--- A value
        Color_HistogramBuffer[i]=sin(i*multiplier);
        //--- Color
        int color_index=i%(bars*color_sections);
        color_index/=bars;
        Color_HistogramColors[i]=color_index;
    }
//--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}

//+-----+
//| Changes the color of line segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
    //--- The number of colors
    int size=ArraySize(cols);
    //---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

    //--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                           PLOT_LINE_COLOR, // Property identifier
                           plot_color_ind, // The index of the color, where we want to change
                           cols[i]); // A new color

        //--- Write the colors
        comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(cols[i]));
        ChartSetString(0,CHART_COMMENT,comm);
    }
}

//+-----+
//| Changes the appearance of a displayed line in the indicator |
//+-----+
void ChangeLineAppearance()
{
    //--- A string for the formation of information about the line properties

```



```
string comm="";  
//--- A block for changing the width of the line  
int number=MathRand();  
//--- Get the width of the remainder of integer division  
int width=number%5; // The width is set from 0 to 4  
//--- Set the color as the PLOT_LINE_WIDTH property  
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);  
//--- Write the line width  
comm=comm+" Width="+IntegerToString(width);  
  
//--- A block for changing the style of the line  
number=MathRand();  
//--- The divisor is equal to the size of the styles array  
int size=ArraySize(styles);  
//--- Get the index to select a new style as the remainder of integer division  
int style_index=number%size;  
//--- Set the color as the PLOT_LINE_COLOR property  
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);  
//--- Write the line style  
comm=EnumToString(styles[style_index])+", "+comm;  
//--- Show the information on the chart using a comment  
Comment(comm);  
}
```


DRAW_COLOR_HISTOGRAM2

The DRAW_COLOR_HISTOGRAM2 style draws a histogram of a specified color - vertical segments using the values of two indicator buffers. But unlike the one-color DRAW_HISTOGRAM2, in this style each column of the histogram can have its own color from a predefined set. The values of all the segments are taken from the indicator buffer.

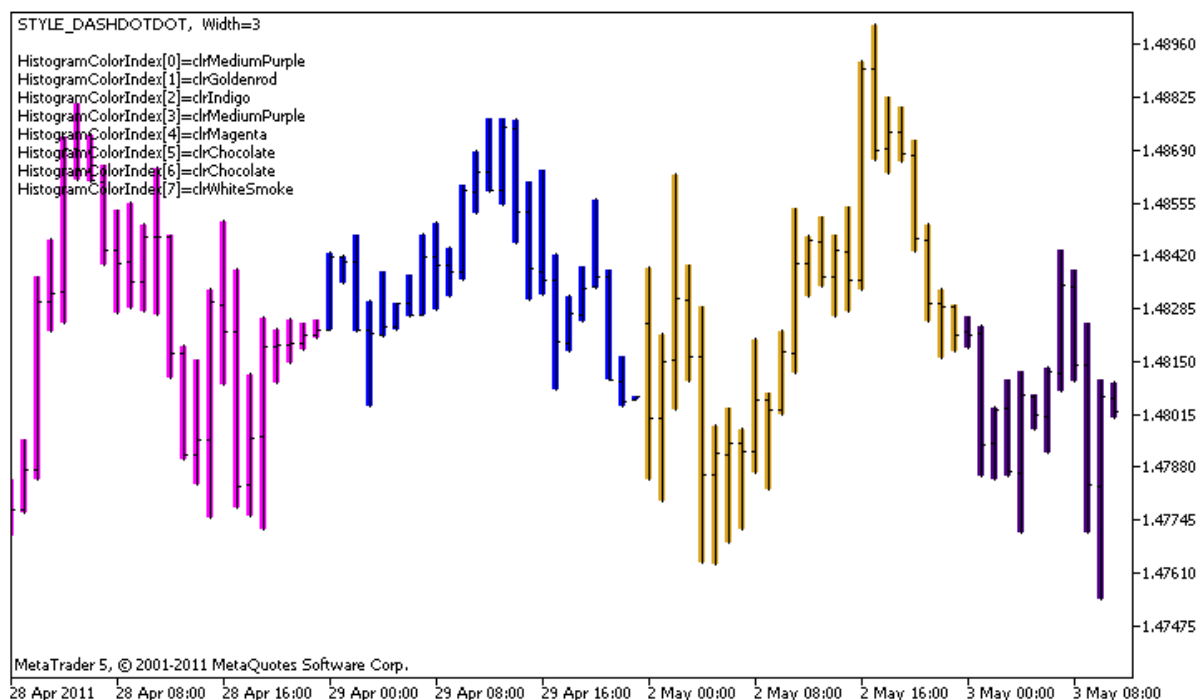
The width, style and color of the histogram can be specified like for the [DRAW_HISTOGRAM2](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

The DRAW_COLOR_HISTOGRAM2 style can be used in a separate subwindow of a chart and in its main window. For empty values nothing is drawn, all the values in the indicator buffers need to be set explicitly. Buffers are not initialized with empty values.

The number of buffers required for plotting DRAW_COLOR_HISTOGRAM2 is 3:

- two buffers to store the upper and lower end of the vertical segment on each bar;
- one buffer to store the color index, which is used to draw the segment (it makes sense to set only non-empty values).

An example of the indicator that draws a histogram of a specified color between the High and Low prices. For each day of week, the histogram lines have a different color. The color of the day, width and style of the histogram change randomly each N ticks.



Please note that for `plot1` with the DRAW_COLOR_HISTOGRAM2 style, 5 colors are set using the compiler directive `#property`, and then in the [OnCalculate\(\)](#) function the colors are selected randomly from the 14 colors stored in the `colors[]` array.

The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).


```

//+-----+
//|                                     DRAW_COLOR_HISTOGRAM2.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 1
//--- plot ColorHistogram_2
#property indicator_label1  "ColorHistogram_2"
#property indicator_type1   DRAW_COLOR_HISTOGRAM2
//--- Define 5 colors for coloring the histogram based on the days of week (they are s
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- input parameter
input int      N=5;           // The number of ticks to change the histogram
int           color_sections;
//--- Value buffers
double        ColorHistogram_2Buffer1[];
double        ColorHistogram_2Buffer2[];
//--- A buffer of color indexes
double        ColorHistogram_2Colors[];
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,ColorHistogram_2Buffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ColorHistogram_2Buffer2,INDICATOR_DATA);

```



```

    SetIndexBuffer(2,ColorHistogram_2Colors,INDICATOR_COLOR_INDEX);
//--- Set an empty value
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- The number of colors to color the sinusoid
    color_sections=8;    // See a comment to #property indicator_color1
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a critical number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Change the colors used to draw the histogram
        ChangeColors(colors,color_sections);
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- Calculate the indicator values
    int start=0;
//--- To get the day of week by the open price of each bar
    MqlDateTime dt;
//--- If already calculated during the previous starts of OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculation
//--- Fill in the indicator buffer with values
    for(int i=start;i<rates_total;i++)
    {
        TimeToStruct(time[i],dt);
        //--- value
        ColorHistogram_2Buffer1[i]=high[i];
        ColorHistogram_2Buffer2[i]=low[i];
    }
}

```



```

        //--- Set the color index according to the day of week
        int day=dt.day_of_week;
        ColorHistogram_2Colors[i]=day;
    }
//--- Return the prev_calculated value for the next call of the function
    return(rates_total);
}
//+-----+
//| Changes the color of line segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- The number of colors
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                            PLOT_LINE_COLOR, // Property identifier
                            plot_color_ind, // The index of the color, where we v
                            cols[i]); // A new color

        //--- Write the colors
        comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToS
        ChartSetString(0,CHART_COMMENT,comm);
    }
//---
}
//+-----+
//| Changes the appearance of a displayed line in the indicator |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the line properties
    string comm="";
//--- A block for changing the width of the line
    int number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width

```



```
comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
number=MathRand();
//--- The divisor is equal to the size of the styles array
int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
Comment(comm);
}
```


DRAW_COLOR_ARROW

The DRAW_COLOR_ARROW style draws colored arrows (symbols of the set [Wingdings](#)) based on the value of the indicator buffer. In contrast to DRAW_ARROW, in this style it is possible to set a color from a predefined set of colors specified by the [indicator_color1](#) property for each symbol.

The width and color of the symbols can be specified like for the [DRAW_ARROW](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows changing the look of an indicator based on the current situation.

The symbol code is set using the [PLOT_ARROW](#) property.

```
//--- Define the symbol code from the Wingdings font to draw in PLOT_ARROW  
PlotIndexSetInteger(0,PLOT_ARROW,code);
```

The default value of PLOT_ARROW=159 (a circle).

Each arrow is actually a symbol that has the height and the anchor point, and can cover some important information on a chart (for example, the closing price at the bar). Therefore, we can additionally specify the vertical shift in pixels, which does not depend on the scale of the chart. The arrows will be shifted down by the specified number of pixels, although the values of the indicator will remain the same:

```
//--- Set the vertical shift of arrows in pixels  
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

A negative value of PLOT_ARROW_SHIFT means the shift of arrows upwards, a positive values shifts the arrow down.

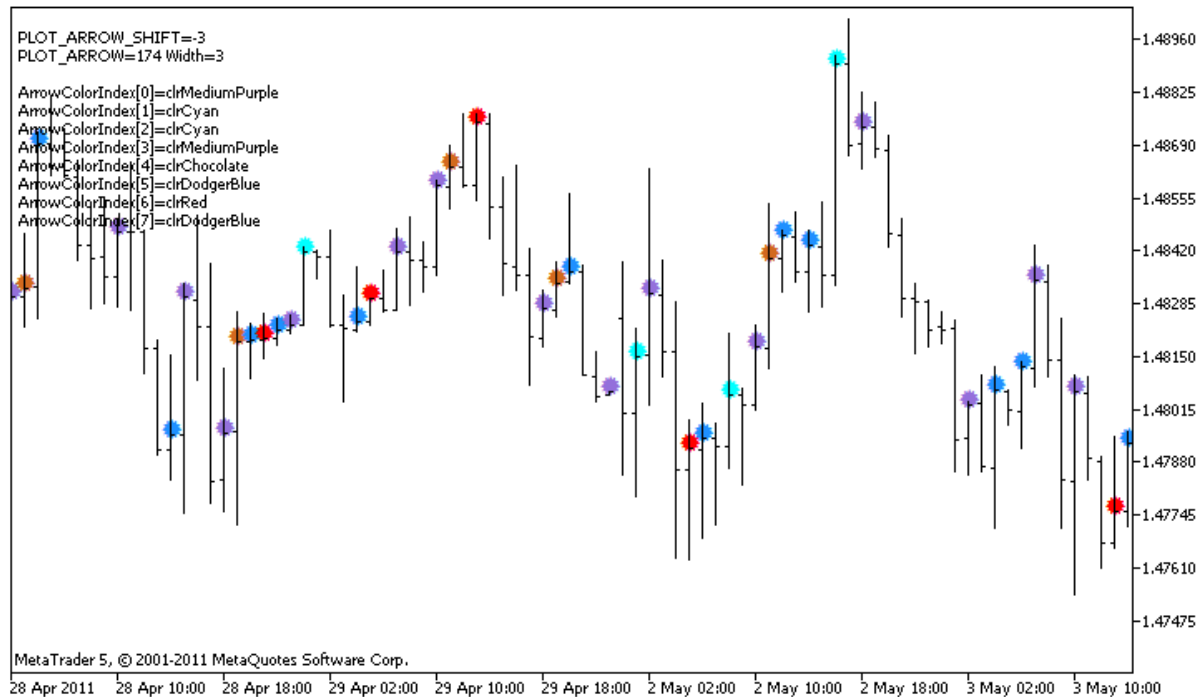
The DRAW_COLOR_ARROW style can be used in a separate subwindow of a chart and in its main window. Empty values are not drawn and do not appear in the "Data Window", all the values in the indicator buffers should be set explicitly. Buffers are not initialized with a zero value.

```
//--- Set an empty value  
PlotIndexSetDouble(DRAW_COLOR_ARROW_plot_index,PLOT_EMPTY_VALUE,0);
```

The number of buffers required for plotting DRAW_COLOR_ARROW is 2.

- a buffer to store the value of the price which is used to draw the symbol (plus a shift in pixels, given in the PLOT_ARROW_SHIFT property);
- a buffer to store the color index, which is used to draw an arrow (it makes sense to set only non-empty values).

An example of the indicator, which draws arrows on each bar with the close price higher than the close price of the previous bar. The width, shift and symbol code of **all** arrows are changed randomly every N ticks. The color of the symbol depends on the number of the bar on which it is drawn.



In the example, for `plot1` with the `DRAW_COLOR_ARROW` style, the properties, color and size are specified using the compiler directive `#property`, and then in the `OnCalculate()` function the properties are set randomly. The `N` parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

Please note that initially 8 colors are set using the compiler directive `#property`, and then in the `OnCalculate()` function, the color is set randomly from the 14 colors that are stored in the `colors[]` array.

```
//+-----+
//|                                     DRAW_COLOR_ARROW.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ARROW"
#property description "Draws different-color arrows set by Unicode characters, on a cl"
#property description "The color, size, shift and symbol code of the arrow are changed"
#property description " randomly every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ColorArrow
#property indicator_label1 "ColorArrow"
#property indicator_type1 DRAW_COLOR_ARROW
```



```

//--- Define 8 colors for coloring the histogram (they are stored in the special array
#property indicator_color1 clrRed,clrBlue,clrSeaGreen,clrGold,clrDarkOrange,clrMagenta
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- input parameters
input int      N=5;           // Number of ticks to change
input ushort   code=159;      // Symbol code to draw in DRAW_ARROW
input          color_sections;
//--- An indicator buffer for the plot
double         ColorArrowBuffer[];
//--- A buffer to store color indexes
double         ColorArrowColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurple;
};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,ColorArrowBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorArrowColors,INDICATOR_COLOR_INDEX);
    //--- Define the symbol code for drawing in PLOT_ARROW
    PlotIndexSetInteger(0,PLOT_ARROW,code);
    //--- Set the vertical shift of arrows in pixels
    PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
    //--- Set as an empty value 0
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
    //--- The number of colors to color the sinusoid
    color_sections=8; // see a comment to #property indicator_color1
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],

```



```

        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
//--- Calculate ticks to change the color, size, shift and code of the arrow
        ticks++;
//--- If a critical number of ticks has been accumulated
        if(ticks>=N)
        {
            //--- Change arrow properties
            ChangeLineAppearance();
            //--- Change the colors used to draw the histogram
            ChangeColors(colors,color_sections);
            //--- Reset the counter of ticks to zero
            ticks=0;
        }

//--- Block for calculating indicator values
        int start=1;
        if(prev_calculated>0) start=prev_calculated-1;
//--- Calculation loop
        for(int i=1;i<rates_total;i++)
        {
            //--- If the current Close price is higher than the previous one, draw an arrow
            if(close[i]>close[i-1])
                ColorArrowBuffer[i]=close[i];
            //--- Otherwise specify the null value
            else
                ColorArrowBuffer[i]=0;
            //--- Arrow color
            int index=i%color_sections;
            ColorArrowColors[i]=index;
        }
//--- return value of prev_calculated for next call
        return(rates_total);
    }
//+-----+
//| Changes the color of line segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- The number of colors
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {

```



```

//--- Get a random value
int number=MathRand();
//--- Get an index in the col[] array as a remainder of the integer division
int i=number%size;
//--- Set the color for each index as the property PLOT_LINE_COLOR
PlotIndexSetInteger(0, // The number of a graphical style
                    PLOT_LINE_COLOR, // Property identifier
                    plot_color_ind, // The index of the color, where we v
                    cols[i]); // A new color

//--- Write the colors
comm=comm+StringFormat("ArrowColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStri
ChartSetString(0,CHART_COMMENT,comm);
}
//---
}
//+-----+
//| Changes the appearance of a displayed line in the indicator |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the line properties
string comm="";
//--- A block for changing the width of the line
int number=MathRand();
//--- Get the width of the remainder of integer division
int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the arrow code (PLOT_ARROW)
number=MathRand();
//--- Get the remainder of integer division to calculate a new code of the arrow (from
int code_add=number%20;
//--- Set the new symbol code as the result of code+code_add
PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- Write the symbol code PLOT_ARROW
comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- A block for changing the vertical shift of arrows in pixels
number=MathRand();
//--- Get the shift as the remainder of the integer division
int shift=20-number%41;
//--- Set the new shift from
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- Write the shift PLOT_ARROW_SHIFT
comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

```



```
//--- Show the information on the chart using a comment  
    Comment(comm);  
}
```


DRAW_COLOR_ZIGZAG

The DRAW_COLOR_ZIGZAG style draws segments of different colors, using the values of two indicator buffers. This style is a colored version of [DRAW_ZIGZAG](#), i.e. allows specifying for each segment an individual color from the predefined set of colors. The segments are plotted from a value in the first buffer to a value in the second indicator buffer. None of the buffers can contain only empty values, since in this case nothing is plotted.

The width, color and style of the line can be specified like for the [DRAW_ZIGZAG](#) style - using [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

Sections are drawn from a non-empty value of one buffer to a non-empty value of another indicator buffer. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

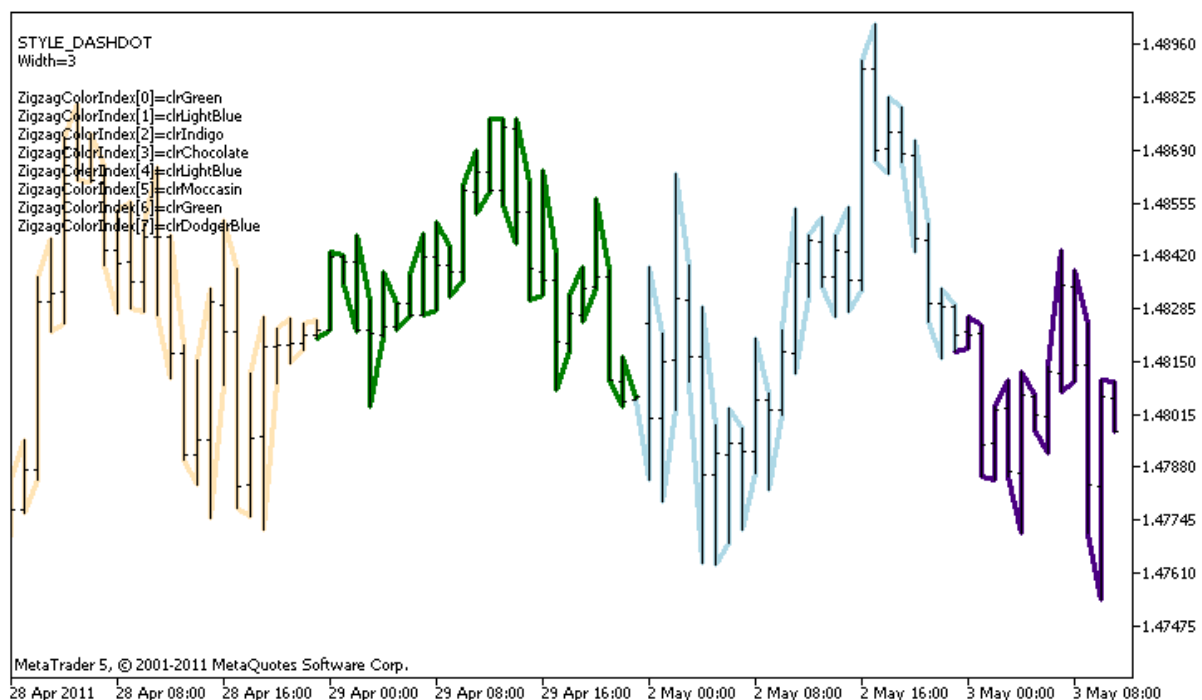
```
//--- The 0 (empty) value will not participate in drawing
PlotIndexSetDouble(index_of_plot_DRAW_COLOR_ZIGZAG, PLOT_EMPTY_VALUE, 0);
```

Always explicitly fill in the of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_COLOR_ZIGZAG is 3:

- two buffers to store the values of ends of the zigzag sections;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

An example of the indicator that plots a saw based on the High and Low prices. The color, width and style of the zigzag lines change randomly every N ticks.



Please note that for `plot1` with the `DRAW_COLOR_ZIGZAG` style, 8 colors are set using the compiler directive `#property`, and then in the `OnCalculate()` function the color is selected randomly from the 14 colors stored in the `colors[]` array.

The `N` parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+-----+
//|                                     DRAW_COLOR_ZIGZAG.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ZIGZAG"
#property description "It draws a broken line as a sequence of colored sections, the c
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 1
//--- plot Color_Zigzag
#property indicator_label1 "Color_Zigzag"
#property indicator_type1  DRAW_COLOR_ZIGZAG
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameter
input int      N=5;           // Number of ticks to change
int          color_sections;
//--- Buffers of values of segment ends
double       Color_ZigzagBuffer1[];
double       Color_ZigzagBuffer2[];
//--- Buffers of color indexes of segment ends
double       Color_ZigzagColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+-----+
//| Custom indicator initialization function |
```



```

//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Color_ZigzagBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,Color_ZigzagBuffer2,INDICATOR_DATA);
    SetIndexBuffer(2,Color_ZigzagColors,INDICATOR_COLOR_INDEX);
//---Number of color for coloring the zigzag
    color_sections=8;    // see a comment to the #property indicator_color1 property
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
    ticks++;
//--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Change colors used to plot the sections
        ChangeColors(colors,color_sections);
        //--- Reset the counter of ticks to zero
        ticks=0;
    }

//--- The structure of time is required to get the day of week of each bar
    MqlDateTime dt;

//--- The start position of calculations
    int start=0;
//--- If the indicator was calculated on the previous tick, then start the calculation
    if(prev_calculated!=0) start=prev_calculated-1;
//--- Calculation loop
    for(int i=start;i<rates_total;i++)

```



```

{
    //--- Write the bar open time in the structure
    TimeToStruct(time[i],dt);

    //--- If the bar number is even
    if(i%2==0)
    {
        //--- Write High in the 1st buffer and Low in the 2nd one
        Color_ZigzagBuffer1[i]=high[i];
        Color_ZigzagBuffer2[i]=low[i];
        //--- The color of the segment
        Color_ZigzagColors[i]=dt.day_of_year%color_sections;
    }
    //--- the bar number is odd
    else
    {
        //--- Fill in the bar in a reverse order
        Color_ZigzagBuffer1[i]=low[i];
        Color_ZigzagBuffer2[i]=high[i];
        //--- The color of the segment
        Color_ZigzagColors[i]=dt.day_of_year%color_sections;
    }
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Changes the color of the zigzag segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
    //--- The number of colors
    int size=ArraySize(cols);
    //---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

    //--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                           PLOT_LINE_COLOR, // Property identifier
                           plot_color_ind, // The index of the color, where we v
                           cols[i]); // A new color

        //--- Write the colors

```



```

        comm=comm+StringFormat("ZigzagColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(plot_color_ind));
        ChartSetString(0,CHART_COMMENT,comm);
    }
//---
}
//+-----+
//| Changes the appearance of the zigzag segments |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the properties of Color_ZigZag
    string comm="";
//--- A block for changing the width of the line
    int number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5;    // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
    number=MathRand();
//--- The divisor is equal to the size of the styles array
    int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
    int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
    PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
    comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}

```


DRAW_COLOR_BARS

The DRAW_COLOR_BARS style draws bars on the values of four indicator buffers, which contain the Open, High, Low and Close prices. This style is an advanced version of [DRAW_BARS](#) and allows specifying for each bar an individual color from the predefined set of colors. It used for creating custom indicators as bars, including those in a separate subwindow of a chart and on other financial instruments.

The color of bars can be set using the [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of all four indicator buffers are set. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

```
//--- The 0 (empty) value will not participate in drawing  
PlotIndexSetDouble(index_of_plot_DRAW_COLOR_BARS,PLOT_EMPTY_VALUE,0);
```

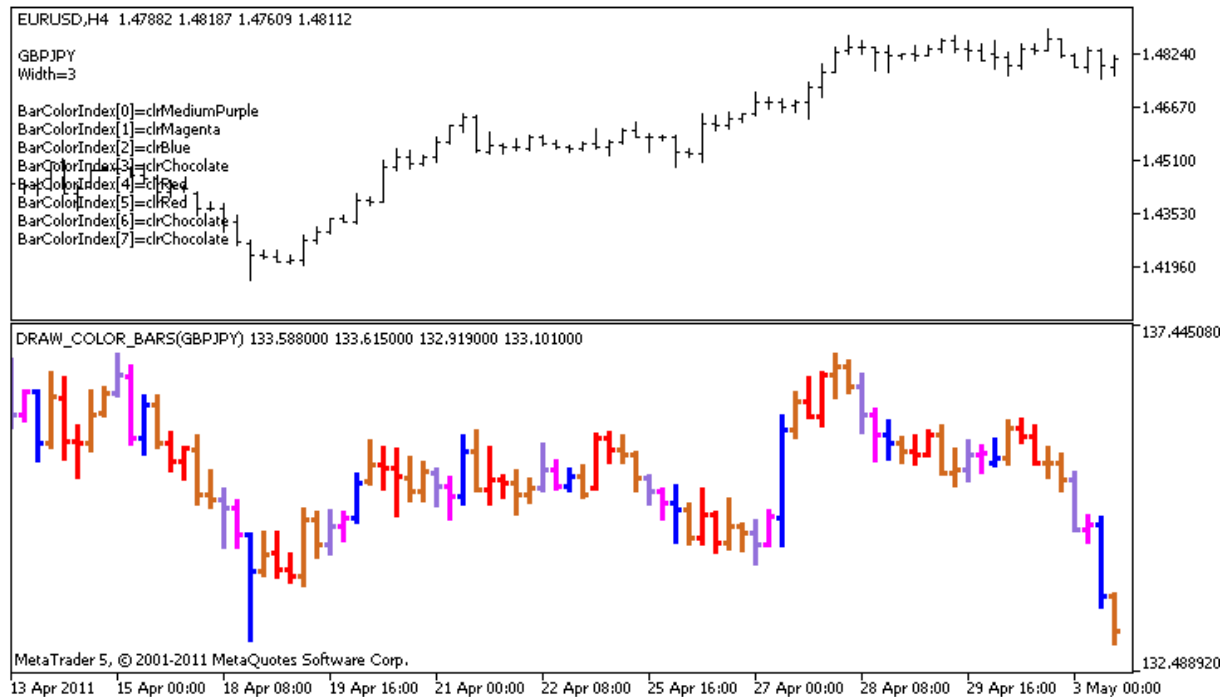
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_COLOR_BARS is 5:

- four buffer to store the values of Open, High, Low and Close;
- one buffer to store the color index, which is used to draw a bar (it makes sense to set it only for the bars that will be drawn).

All buffers for the plotting should go one after the other in the given order: Open, High, Low, Close and the color buffer. None of the price buffers can contain only null values, since in this case nothing is plotted.

An example of the indicator that draws bars on a selected financial instrument in a separate window. The color of bars changes randomly every N ticks. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for `plot1` with the `DRAW_COLOR_BARS` style, 8 colors are set using the compiler directive `#property`, and then in the `OnCalculate()` function the color is selected randomly from the 14 colors stored in the `colors[]` array.

```
//+-----+
//|                                     DRAW_COLOR_BARS.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_BARS"
#property description "It draws different-color bars of a selected symbol in a separate window"
#property description "The color and width of bars, as well as the symbol are changed every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots 1
//--- plot ColorBars
#property indicator_label1 "ColorBars"
#property indicator_type1  DRAW_COLOR_BARS
//--- Define 8 colors for coloring bars (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLightBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;          // The number of ticks to change the type
```



```

input int      bars=500;          // The number of bars to show
input bool     messages=false;    // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double        ColorBarsBuffer1[];
double        ColorBarsBuffer2[];
double        ColorBarsBuffer3[];
double        ColorBarsBuffer4[];
double        ColorBarsColors[];
//--- Symbol name
string symbol;
int    bars_colors;
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,ColorBarsBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ColorBarsBuffer2,INDICATOR_DATA);
    SetIndexBuffer(2,ColorBarsBuffer3,INDICATOR_DATA);
    SetIndexBuffer(3,ColorBarsBuffer4,INDICATOR_DATA);
    SetIndexBuffer(4,ColorBarsColors,INDICATOR_COLOR_INDEX);
    //--- Number of colors for coloring bars
    bars_colors=8;    // see a comment to the #property indicator_color1 property
    //---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
    //--- Count ticks to change the style, color and width of the bar

```



```

    ticks++;
//--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Select a new symbol from the Market watch window
        symbol=GetRandomSymbolName();
        //--- Change the line properties
        ChangeLineAppearance();
        //--- Change the colors used to draw the candlesticks
        ChangeColors(colors,bars_colors);
        int tries=0;
        //--- Make 5 attempts to fill in the buffers with the prices from symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total,bars_colors) && tries<5)
        {
            //--- A counter of calls of the CopyFromSymbolToBuffers() function
            tries++;
        }
        //--- Reset the counter of ticks to zero
        ticks=0;
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Fill in the indicator buffers with prices |
//+-----+
bool CopyFromSymbolToBuffers(string name,int total,int bar_colors)
{
    //--- In the rates[] array, we will copy Open, High, Low and Close
    MqlRates rates[];
    //--- The counter of attempts
    int attempts=0;
    //--- How much has been copied
    int copied=0;
    //--- Make 25 attempts to get a timeseries on the desired symbol
    while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
    {
        Sleep(100);
        attempts++;
        if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
    }
    //--- If failed to copy a sufficient number of bars
    if(copied!=bars)
    {
        //--- Form a message string
        string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                                name,
                                copied,
                                bars

```



```

        );

    //--- Show a message in a comment in the main chart window
    Comment(comm);
    //--- Show the message
    if(messages) Print(comm);
    return(false);
}
else
{
    //--- Set the display of the symbol
    PlotIndexSetString(0,PLOT_LABEL,name+" Open;" +name+" High;" +name+" Low;" +name+"
    IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_BARS(" +name+" )");
}

//--- Initialize buffers with empty values
ArrayInitialize(ColorBarsBuffer1,0.0);
ArrayInitialize(ColorBarsBuffer2,0.0);
ArrayInitialize(ColorBarsBuffer3,0.0);
ArrayInitialize(ColorBarsBuffer4,0.0);

//--- Copy prices to the buffers
for(int i=0;i<copied;i++)
{
    //--- Calculate the appropriate index for the buffers
    int buffer_index=total-copied+i;
    //--- Write the prices to the buffers
    ColorBarsBuffer1[buffer_index]=rates[i].open;
    ColorBarsBuffer2[buffer_index]=rates[i].high;
    ColorBarsBuffer3[buffer_index]=rates[i].low;
    ColorBarsBuffer4[buffer_index]=rates[i].close;
    //---
    ColorBarsColors[buffer_index]=i%bar_colors;
}
return(true);
}

//+-----+
//| Randomly returns a symbol from the Market Watch |
//+-----+
string GetRandomSymbolName()
{
    //--- The number of symbols shown in the Market watch window
    int symbols=SymbolsTotal(true);
    //--- The position of a symbol in the list - a random number from 0 to symbols
    int number=MathRand()%symbols;
    //--- Return the name of a symbol at the specified position
    return SymbolName(number,true);
}

//+-----+
//| Changes the color of the zigzag segments |
//+-----+

```



```

void ChangeColors(color &cols[],int plot_colors)
{
//--- The number of colors
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                            PLOT_LINE_COLOR, // Property identifier
                            plot_color_ind, // The index of the color, where we v
                            cols[i]); // A new color

        //--- Write the colors
        comm=comm+StringFormat("BarColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
        ChartSetString(0,CHART_COMMENT,comm);
    }
//---
}

//+-----+
//| Changes the appearance of bars |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the bar properties
    string comm="";

//--- A block for changing the width of bars
    int number=MathRand();
//--- Get the width of the remainder of integer division
    int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
    comm=comm+"\r\nWidth="+IntegerToString(width);

//--- Write the symbol name
    comm="\r\n"+symbol+comm;

//--- Show the information on the chart using a comment
    Comment(comm);
}

```


DRAW_COLOR_CANDLES

The DRAW_COLOR_CANDLES style, like [DRAW_CANDLES](#), draws candlesticks using the values of four indicator buffers, which contain Open, High, Low and Close prices. In addition, it allows specifying a color for each candlestick from a given set. For this purpose, the style has a special color buffer that stores color indexes for each bar. It used for creating custom indicators as a sequence of candlesticks, including those in a separate subwindow of a chart and on other financial instruments.

The number of colors of candlesticks can be set using the [compiler directives](#) or dynamically using the [PlotIndexSetInteger\(\)](#) function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of four price buffers of the indicator are set. To specify what value should be considered as "empty", set this value in the [PLOT_EMPTY_VALUE](#) property:

```
//--- The 0 (empty) value will not participate in drawing  
PlotIndexSetDouble(index_of_plot_DRAW_COLOR_CANDLES, PLOT_EMPTY_VALUE, 0);
```

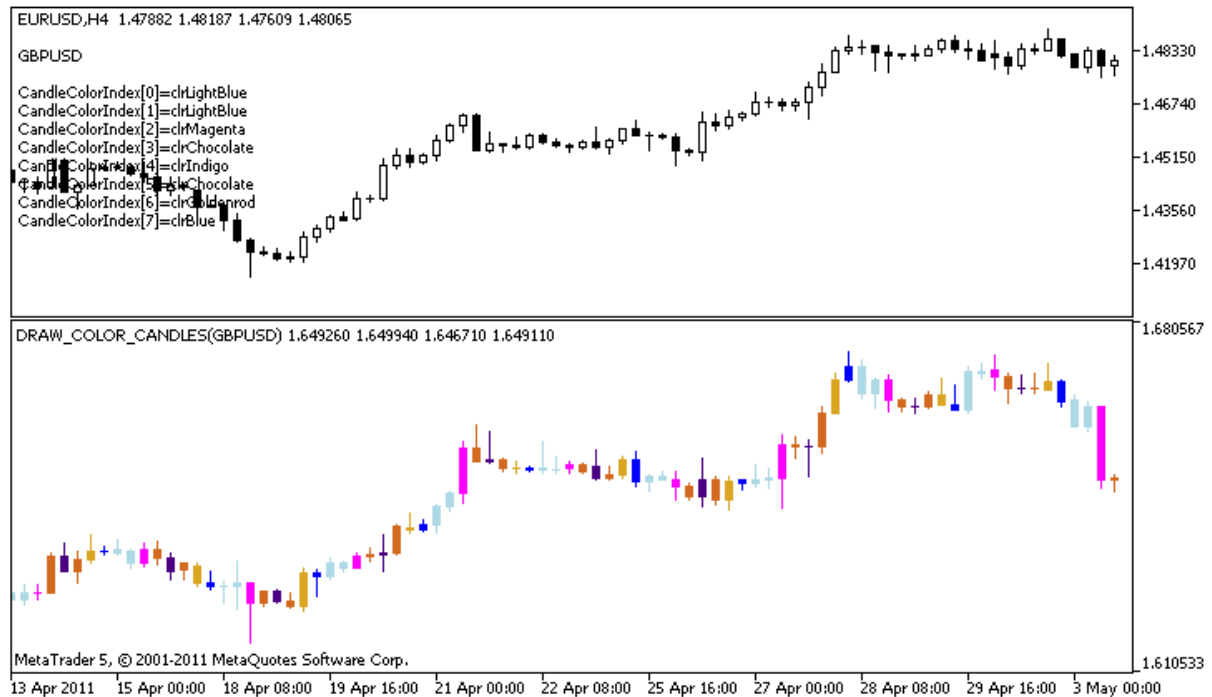
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_COLOR_CANDLES is 5:

- four buffer to store the values of Open, High, Low and Close;
- one buffer to store the color index, which is used to draw a candlestick (it makes sense to set it only for the candlesticks that will be drawn).

All buffers for the plotting should go one after the other in the given order: Open, High, Low, Close and the color buffer. None of the price buffers can contain only empty values, since in this case nothing is plotted.

An example of the indicator that draws candlesticks for a selected financial instrument in a separate window. The color of candlesticks changes randomly every N ticks. The N parameter is set in [external parameters](#) of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for `plot1`, the color is set using the compiler directive `#property`, and then in the `OnCalculate()` function the color is set randomly from an earlier prepared list.

```
//+-----+
//|                                     DRAW_COLOR_CANDLES.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate window"
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are"
#property description "randomly every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots 1
//--- plot ColorCandles
#property indicator_label1 "ColorCandles"
#property indicator_type1 DRAW_COLOR_CANDLES
//--- Define 8 colors for coloring candlesticks (they are stored in the special array)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLightBlue,clrLightGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
```



```

//--- input parameters
input int      N=5;           // The number of ticks to change the type
input int      bars=500;      // The number of candlesticks to show
input bool     messages=false; // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double        ColorCandlesBuffer1[];
double        ColorCandlesBuffer2[];
double        ColorCandlesBuffer3[];
double        ColorCandlesBuffer4[];
double        ColorCandlesColors[];
int           candles_colors;
//--- Symbol name
string symbol;
//--- An array for storing colors contains 14 elements
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- If bars is very small - complete the work ahead of time
    if(bars<50)
    {
        Comment("Please specify a larger number of bars! The operation of the indicator
        return(INIT_PARAMETERS_INCORRECT);
    }
    //--- indicator buffers mapping
    SetIndexBuffer(0,ColorCandlesBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ColorCandlesBuffer2,INDICATOR_DATA);
    SetIndexBuffer(2,ColorCandlesBuffer3,INDICATOR_DATA);
    SetIndexBuffer(3,ColorCandlesBuffer4,INDICATOR_DATA);
    SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
    //--- An empty value
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
    //--- The name of the symbol, for which the bars are drawn
    symbol=_Symbol;
    //--- Set the display of the symbol
    PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symbol+" Close;");
    IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
    //--- The number of colors to color candlesticks
    candles_colors=8; // see. a comment to the #property indicator_color1 property
    //---
    return(INIT_SUCCEEDED);
}
//+-----+

```



```

//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=INT_MAX-100;
    //--- Count ticks to change the style and color
    ticks++;
    //--- If a sufficient number of ticks has been accumulated
    if(ticks>=N)
    {
        //--- Select a new symbol from the Market watch window
        symbol=GetRandomSymbolName();
        //--- Change the form
        ChangeLineAppearance();
        //--- Change the colors used to draw the candlesticks
        ChangeColors(colors,candles_colors);

        int tries=0;
        //--- Make 5 attempts to fill in the buffers of plot1 with the prices from symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
                                       ColorCandlesBuffer1,ColorCandlesBuffer2,ColorCandlesBuffer3,
                                       ColorCandlesBuffer4,ColorCandlesColors,candles_colors)
              && tries<5)
        {
            //--- A counter of calls of the CopyFromSymbolToBuffers() function
            tries++;
        }
        //--- Reset the counter of ticks to zero
        ticks=0;
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Fills in the specified candlestick |
//+-----+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],

```



```

        double &buff2[],
        double &buff3[],
        double &buff4[],
        double &col_buffer[],
        int cndl_colors
    )

{
//--- In the rates[] array, we will copy Open, High, Low and Close
    MqlRates rates[];
//--- The counter of attempts
    int attempts=0;
//--- How much has been copied
    int copied=0;
//--- Make 25 attempts to get a timeseries on the desired symbol
    while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
    {
        Sleep(100);
        attempts++;
        if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
    }
//--- If failed to copy a sufficient number of bars
    if(copied!=bars)
    {
        //--- Form a message string
        string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                                name,
                                copied,
                                bars
                                );

        //--- Show a message in a comment in the main chart window
        Comment(comm);
        //--- Show the message
        if(messages) Print(comm);
        return(false);
    }
    else
    {
        //--- Set the display of the symbol
        PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;" +name+" High;" +name+" Low;
        IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES (" +symbol+" )");
    }
//--- Initialize buffers with empty values
    ArrayInitialize(buff1,0.0);
    ArrayInitialize(buff2,0.0);
    ArrayInitialize(buff3,0.0);
    ArrayInitialize(buff4,0.0);
//--- On each tick copy prices to buffers
    for(int i=0;i<copied;i++)
    {

```



```

    //--- Calculate the appropriate index for the buffers
    int buffer_index=total-copied+i;
    //--- Write the prices to the buffers
    buff1[buffer_index]=rates[i].open;
    buff2[buffer_index]=rates[i].high;
    buff3[buffer_index]=rates[i].low;
    buff4[buffer_index]=rates[i].close;
    //--- Set the candlestick color
    int color_index=i%cndl_colors;
    col_buffer[buffer_index]=color_index;
}
return(true);
}

//+-----+
//| Randomly returns a symbol from the Market Watch |
//+-----+
string GetRandomSymbolName()
{
    //--- The number of symbols shown in the Market watch window
    int symbols=SymbolsTotal(true);
    //--- The position of a symbol in the list - a random number from 0 to symbols
    int number=MathRand()%symbols;
    //--- Return the name of a symbol at the specified position
    return SymbolName(number,true);
}

//+-----+
//| Changes the color of the candlestick segments |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
    //--- The number of colors
    int size=ArraySize(cols);
    //---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

    //--- For each color index define a new color randomly
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
        //--- Get a random value
        int number=MathRand();
        //--- Get an index in the col[] array as a remainder of the integer division
        int i=number%size;
        //--- Set the color for each index as the property PLOT_LINE_COLOR
        PlotIndexSetInteger(0, // The number of a graphical style
                           PLOT_LINE_COLOR, // Property identifier
                           plot_color_ind, // The index of the color, where we v
                           cols[i]); // A new color

        //--- Write the colors
        comm=comm+StringFormat("CandleColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr

```



```
        ChartSetString(0, CHART_COMMENT, comm);
    }
//---
}
//+-----+
//| Changes the appearance of candlesticks |
//+-----+
void ChangeLineAppearance()
{
//--- A string for the formation of information about the candlestick properties
    string comm="";
//--- Write the symbol name
    comm="\r\n"+symbol+comm;
//--- Show the information on the chart using a comment
    Comment(comm);
}
```


Connection between Indicator Properties and Corresponding Functions

Every custom indicator has numerous [properties](#), some of which are obligatory and are always positioned at the beginning of description. They are the following properties:

- indication of a window to plot the indicator - `indicator_separate_window` or `indicator_chart_window`;
- number of indicator buffers - `indicator_buffers`;
- number of plots of the indicator - `indicator_plots`.

Also there are other properties that can be set both through [preprocessor](#) directives and through functions intended for custom indicator creation. These properties and corresponding functions are described in the following table.

Directives for properties of indicator subwindow	Functions of IndicatorSet... () type	Description of the adjusted property of the subwindow
<code>indicator_height</code>	IndicatorSetInteger (INDICATOR_INDICATOR_HEIGHT , nHeight)	The fixed value of the subwindow height
<code>indicator_minimum</code>	IndicatorSetDouble (INDICATOR_MINIMUM , dMaxValue)	Minimal value of the vertical axis
<code>indicator_maximum</code>	IndicatorSetDouble (INDICATOR_MAXIMUM , dMinValue)	Maximal value of the vertical axis
<code>indicator_levelN</code>	IndicatorSetDouble (INDICATOR_LEVELVALUE , N-1, nLevelValue)	Vertical axis value for N level
no preprocessor directive	IndicatorSetString (INDICATOR_LEVELTEXT , N-1, sLevelName)	Name of a displayed level
<code>indicator_levelcolor</code>	IndicatorSetInteger (INDICATOR_LEVELCOLOR , N-1, nLevelColor)	Color of N level
<code>indicator_levelwidth</code>	IndicatorSetInteger (INDICATOR_LEVELWIDTH , N-1, nLevelWidth)	Line width for N level
<code>indicator_levelstyle</code>	IndicatorSetInteger (INDICATOR_LEVELSTYLE , N-1, nLevelStyle)	Line style for N level
Directives for plotting properties	Functions of PlotIndexSet... () type	Description of the adjusted property of a plot
<code>indicator_labelN</code>	PlotIndexSetString (N-1, PLOT_LABEL ,sLabel)	Short name of the number N plot. It is displayed in

		DataWindow and in the pop-up tooltip when pointing the mouse cursor over it
indicator_colorN	PlotIndexSetInteger (N-1, PLOT_LINE_COLOR , nColor)	Line color for N plot
indicator_styleN	PlotIndexSetInteger (N-1, PLOT_LINE_STYLE , nType)	Line style for N plot
indicator_typeN	PlotIndexSetInteger (N-1, PLOT_DRAW_TYPE , nType)	Line type for N plot
indicator_widthN	PlotIndexSetInteger (N-1, PLOT_LINE_WIDTH , nWidth)	Line width for N plot
Common indicator properties	Functions of IndicatorSet... () type	Description
no preprocessor directive	IndicatorSetString (INDICATOR_SHORTNAME , sShortName)	Sets the convenient short name of the indicator that will be displayed in the list of indicators (opened in the terminal by pressing Ctrl+I).
no preprocessor directive	IndicatorSetInteger (INDICATOR_DIGITS , nDigits)	Sets required accuracy of display of indicator values - number of decimal places
no preprocessor directive	IndicatorSetInteger (INDICATOR_LEVELS , nLevels)	Sets number of levels in the indicator window
indicator_applied_price	No function, the property can be set only by the preprocessor directive.	Default price type used for the indicator calculation. It is specified when necessary, only if OnCalculate() of the first type is used. The property value can also be set from the dialog of indicator properties in the "Parameters" tab - "Apply to" .

It should be noted that numeration of levels and plots in preprocessor terms starts with one, while numeration of the same properties at using functions starts with zero, i.e. the indicated value must be by 1 less than that indicated for #property.

There are several directives, for which there are no corresponding functions:

Directive	Description
indicator_chart_window	Indicator is displayed in the main window
indicator_separate_window	Indicator is displayed in a separate subwindow
indicator_buffers	Number of required indicator buffers

indicator_plots

Number of [plots](#) in the indicator

SetIndexBuffer

The function binds a specified indicator buffer with one-dimensional dynamic array of the [double](#) type.

```
bool SetIndexBuffer(
    int          index,          // buffer index
    double       buffer[],       // array
    ENUM_INDEXBUFFER_TYPE data_type // what will be stored
);
```

Parameters

index

[in] Number of the indicator buffer. The numbering starts with 0. The number must be less than the value declared in [#property indicator_buffers](#).

buffer[]

[in] An array declared in the custom indicator program.

data_type

[in] Type of data stored in the indicator array. By default it is [INDICATOR_DATA](#) (values of the calculated indicator). It may also take the value of [INDICATOR_COLOR_INDEX](#); in this case this buffer is used for storing color indexes for the previous indicator buffer. You can specify up to 64 [colors](#) in the [#property indicator_colorN](#) line. The [INDICATOR_CALCULATIONS](#) value means that the buffer is used in intermediate calculations of the indicator and is not intended for drawing.

Return Value

If successful, returns [true](#), otherwise - [false](#).

Note

After binding, the dynamic array `buffer[]` will be indexed as in common arrays, even if the indexing of [timeseries](#) is pre-installed for the bound array. If you want to change the order of access to elements of the indicator array, use the [ArraySetAsSeries\(\)](#) function **after binding the array using the `SetIndexBuffer()` function**. Please note that you can't change the size for dynamic arrays set as indicator buffers by the function [SetIndexBuffer\(\)](#). For indicator buffers, all operations of size changes are performed by the executing sub-system of the terminal.

Example:

```
//+-----+
//|                                     TestCopyBuffer1.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
```



```

//---- plot MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool                  AsSeries=true;
input int                  period=15;
input ENUM_MA_METHOD       smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE   price=PRICE_CLOSE;
input int                  shift=0;
//--- indicator buffers
double                     MABuffer[];
int                         ma_handle;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    if(AsSeries) ArraySetAsSeries(MABuffer,true);
    Print("Indicator buffer is timeseries = ",ArrayGetAsSeries(MABuffer));
    SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
    Print("Indicator buffer after SetIndexBuffer() is timeseries = ",
          ArrayGetAsSeries(MABuffer));

//--- change the order of accessing elements of the indicator buffer
    ArraySetAsSeries(MABuffer,AsSeries);

    IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//---
    ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{

```



```
//--- Copy the values of the moving average in the buffer MABuffer
int copied=CopyBuffer(ma_handle,0,0,rates_total,MABuffer);

Print("MABuffer[0] = ",MABuffer[0]); // Depending on the value AsSeries
                                     // Will receive a very old value
                                     // Or for the current unfinished bar

//--- return value of prev_calculated for next call
return(rates_total);
}
//+-----+
```

See also

[Custom Indicator Properties](#), [Access to timeseries and indicators](#)

IndicatorSetDouble

The function sets the value of the corresponding indicator property. Indicator property must be of the double type. There are two variants of the function.

Call with specifying the property identifier.

```
bool IndicatorSetDouble(
    int    prop_id,           // identifier
    double prop_value        // value to be set
);
```

Call with specifying the property identifier and modifier.

```
bool IndicatorSetDouble(
    int    prop_id,           // identifier
    int    prop_modifier,    // modifier
    double prop_value        // value to be set
)
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the [ENUM_CUSTOMIND_PROPERTY_DOUBLE](#) enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier. Numbering of levels starts from 0. It means that in order to set property for the second level you need to specify 1 (1 less than when using [compiler directive](#)).

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, [indicator display](#) can differ from the intended one.

For example, the first level value for the indicator in a separate subwindow can be set in two ways:

- property indicator_level1 50 - the value of 1 is used for specifying the level number,
- IndicatorSetDouble(INDICATOR_LEVELVALUE, 0, 50) - 0 is used for specifying the first level.

Example: indicator that turns upside down the maximum and minimum values of the indicator window and values of levels on which the horizontal lines are placed.



```
#property indicator_separate_window
//--- set the maximum and minimum values for the indicator window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
//--- set thickness of horizontal levels
#property indicator_levelwidth 1
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- set descriptions of horizontal levels
IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- set color for each level
IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,clrGreen);
IndicatorSetInteger(INDICATOR_LEVELCOLOR,2,clrRed);
```



```

//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int tick_counter=0;
    static double level1=25,level2=50,level3=75;
    static double max=100,min=0, shift=100;
//--- calculate ticks
    tick_counter++;
//--- turn levels upside down on every 10th tick
    if(tick_counter%10==0)
    {
        //--- invert sign for the level values
        level1=-level1;
        level2=-level2;
        level3=-level3;
        //--- invert sign for the maximum and minimum values
        max-=shift;
        min-=shift;
        //--- invert the shift value
        shift=-shift;
        //--- set new level values
        IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
        IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
        IndicatorSetDouble(INDICATOR_LEVELVALUE,2,level3);
        //--- set new values of maximum and minimum in the indicator window
        Print("Set up max = ",max," min = ",min);
        IndicatorSetDouble(INDICATOR_MAXIMUM,max);
        IndicatorSetDouble(INDICATOR_MINIMUM,min);
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

See also

[Indicator Styles in Examples](#), [Connection between Indicator Properties and Functions](#), [Drawing Styles](#)

IndicatorSetInteger

The function sets the value of the corresponding indicator property. Indicator property must be of the int or color type. There are two variants of the function.

Call with specifying the property identifier.

```
bool IndicatorSetInteger (
    int prop_id,           // identifier
    int prop_value         // value to be set
);
```

Call with specifying the property identifier and modifier.

```
bool IndicatorSetInteger (
    int prop_id,           // identifier
    int prop_modifier,     // modifier
    int prop_value         // value to be set
);
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the [ENUM_CUSTOMIND_PROPERTY_INTEGER](#) enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

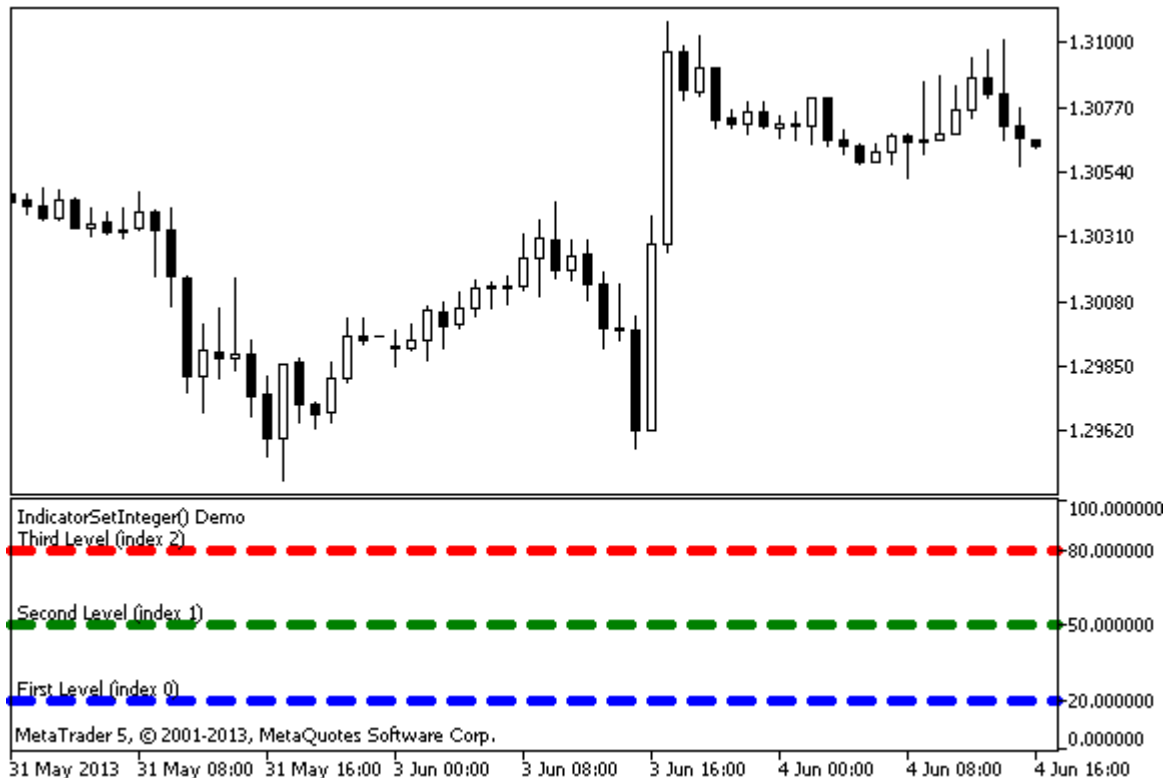
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, [indicator display](#) can differ from the intended one.

For example, in order to set thickness of the first horizontal line use zeroth index:

- `IndicatorSetInteger(INDICATOR_LEVELWIDTH, 0, 5)` - index 0 is used to set thickness of the first level.

Example: indicator that sets color, style and thickness of the indicator horizontal lines.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- set thickness of horizontal levels
#property indicator_levelwidth 5
//--- set color of horizontal levels
#property indicator_levelcolor clrAliceBlue
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- set descriptions of horizontal levels
IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetInteger() Demo");
return(INIT_SUCCEEDED);
}
//+-----+
```



```

//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int tick_counter=0;
//--- calculate ticks
    tick_counter++;
//--- and calculate colors of horizontal levels depending on the tick counter
    ChangeLevelColor(0,tick_counter,3,6,10); // three last parameters are switching the
    ChangeLevelColor(1,tick_counter,3,6,8);
    ChangeLevelColor(2,tick_counter,4,7,9);
//--- modify style of horizontal levels
    ChangeLevelStyle(0,tick_counter);
    ChangeLevelStyle(1,tick_counter+5);
    ChangeLevelStyle(2,tick_counter+15);
//--- get width as the remainder of integer division of the ticks number by 5
    int width=tick_counter%5;
//--- iterate over all horizontal levels and set thickness
    for(int l=0;l<3;l++)
        IndicatorSetInteger(INDICATOR_LEVELWIDTH,l,width+1);
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Set color of horizontal line in the separate indicator window |
//+-----+
void ChangeLevelColor(int level, // number of horizontal line
                     int tick_number, // dividend, number to get the remainder of divi
                     int f_trigger, // first divisor of color switching
                     int s_trigger, // second divisor of color switching
                     int t_trigger) // third divisor of color switching
{
    static color colors[3]={clrRed,clrBlue,clrGreen};
//--- index of color from the colors[] array
    int index=-1;
//--- calculate the number of color from the colors[] array to paint horizontal line
    if(tick_number%f_trigger==0)
        index=0; // if tick_number is divided by f_trigger without the remainder
    if(tick_number%s_trigger==0)
        index=1; // if tick_number is divided by s_trigger without the remainder

```



```

    if(tick_number%t_trigger==0)
        index=2;    // if tick_number is divided by t_trigger without the remainder
//--- if color is defined, set it
    if(index!=-1)
        IndicatorSetInteger(INDICATOR_LEVELCOLOR,level,colors[index]);
//---
}
//+-----+
//| Set style of horizontal line in the separate indicator window |
//+-----+
void ChangeLevelStyle(int level,    // number of horizontal line
                     int tick_number// number to get the remainder of division
                     )
{
//--- array to store styles
    static ENUM_LINE_STYLE styles[5]=
        {STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- index of style from the styles[] array
    int index=-1;
//--- calculate the number from the styles[] array to set style of horizontal line
    if(tick_number%50==0)
        index=5;    // if tick_number is divided by 50 without the remainder, then style
    if(tick_number%40==0)
        index=4;    // ... style is STYLE_DASHDOT
    if(tick_number%30==0)
        index=3;    // ... STYLE_DOT
    if(tick_number%20==0)
        index=2;    // ... STYLE_DASH
    if(tick_number%10==0)
        index=1;    // ... STYLE_SOLID
//--- if style is defined, set it
    if(index!=-1)
        IndicatorSetInteger(INDICATOR_LEVELSTYLE,level,styles[index]);
}

```

See also

[Custom Indicator Properties](#), [Program Properties \(#property\)](#), [Drawing Styles](#)

IndicatorSetString

The function sets the value of the corresponding indicator property. Indicator property must be of the string type. There are two variants of the function.

Call with specifying the property identifier.

```
bool IndicatorSetString(
    int     prop_id,           // identifier
    string  prop_value        // value to be set
);
```

Call with specifying the property identifier and modifier.

```
bool IndicatorSetString(
    int     prop_id,           // identifier
    int     prop_modifier,    // modifier
    string  prop_value        // value to be set
)
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the [ENUM_CUSTOMIND_PROPERTY_STRING](#) enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, [indicator display](#) can differ from the intended one.

For example, in order to set description of the first horizontal line use zeroth index:

- IndicatorSetString(INDICATOR_LEVELTEXT, 0, "First Level") - index 0 is used to set text description of the first level.

Example: indicator that sets text labels to the indicator horizontal lines.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- set color of horizontal levels
#property indicator_levelcolor clrRed
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_SOLID
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- set descriptions of horizontal levels
IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetString() Demo");
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
```



```
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//---

//--- return value of prev_calculated for next call
    return(rates_total);
}
```

See also

[Custom Indicator Properties](#), [Program Properties \(#property\)](#)

PlotIndexSetDouble

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the double type.

```
bool PlotIndexSetDouble(  
    int     plot_index,    // plotting style index  
    int     prop_id,       // property identifier  
    double  prop_value     // value to be set  
);
```

Parameters

plot_index

[in] Index of the [graphical plotting](#)

prop_id

[in] The value can be one of the values of the [ENUM_PLOT_PROPERTY_DOUBLE](#) enumeration.

prop_value

[in] The value of the property.

Return Value

If successful, returns [true](#), otherwise [false](#).

PlotIndexSetInteger

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the int, char, bool or color type. There are 2 variants of the function.

Call indicating identifier of the property.

```
bool PlotIndexSetInteger (
    int  plot_index,      // plotting style index
    int  prop_id,         // property identifier
    int  prop_value       // value to be set
);
```

Call indicating the identifier and modifier of the property.

```
bool PlotIndexSetInteger (
    int  plot_index,      // plotting style index
    int  prop_id,         // property identifier
    int  prop_modifier,   // property modifier
    int  prop_value       // value to be set
)
```

Parameters

plot_index

[in] Index of the [graphical plotting](#)

prop_id

[in] The value can be one of the values of the [ENUM_PLOT_PROPERTY_INTEGER](#) enumeration.

prop_modifier

[in] Modifier of the specified property. Only color index properties require a modifier.

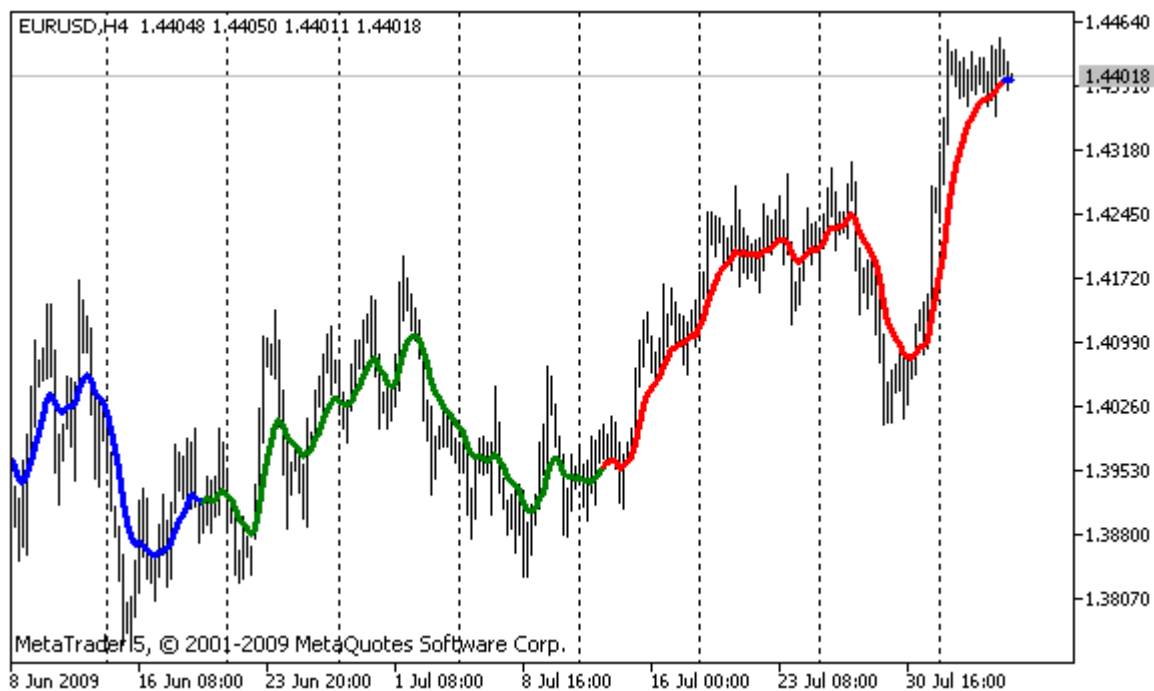
prop_value

[in] The value of the property.

Return Value

If successful, returns [true](#), otherwise [false](#).

Example: an indicator that draws a three-color line. The color scheme changes every 5 ticks.



```
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//---- plot ColorLine
#property indicator_label1 "ColorLine"
#property indicator_type1 DRAW_COLOR_LINE
#property indicator_color1 clrRed,clrGreen,clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 3
//--- indicator buffers
double ColorLineBuffer[];
double ColorBuffer[];
int MA_handle;
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
MA_handle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
//---
}
//+-----+
//| get color index |
//+-----+
```



```

int getIndexOfColor(int i)
{
    int j=i%300;
    if(j<100) return(0); // first index
    if(j<200) return(1); // second index
    return(2); // third index
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //---
    static int ticks=0,modified=0;
    int limit;
    //--- first calculation or number of bars was changed
    if(prev_calculated==0)
    {
        //--- copy values of MA into indicator buffer ColorLineBuffer
        int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
        if(copied<=0) return(0); // copying failed - throw away
        //--- now set line color for every bar
        for(int i=0;i<rates_total;i++)
            ColorBuffer[i]=getIndexOfColor(i);
    }
    else
    {
        //--- copy values of MA into indicator buffer ColorLineBuffer
        int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
        if(copied<=0) return(0);

        ticks++; // ticks counting
        if(ticks>=5) // it's time to change color scheme
        {
            ticks=0; // reset counter
            modified++; // counter of color changes
            if(modified>=3) modified=0; // reset counter
            ResetLastError();
            switch(modified)
            {

```



```

    case 0:// first color scheme
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrRed);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrBlue);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrGreen);
        Print("Color scheme "+modified);
        break;
    case 1:// second color scheme
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrYellow);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrPink);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLightSlateGray);
        Print("Color scheme "+modified);
        break;
    default:// third color scheme
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrLightGoldenrod);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrOrchid);
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLimeGreen);
        Print("Color scheme "+modified);
    }
}
else
{
    //--- set start position
    limit=prev_calculated-1;
    //--- now we set line color for every bar
    for(int i=limit;i<rates_total;i++)
        ColorBuffer[i]=getIndexOfColor(i);
}

//--- return value of prev_calculated for next call
return(rates_total);
}
//+-----+

```


PlotIndexSetString

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the string type.

```
bool PlotIndexSetString(  
    int    plot_index,    // plotting style index  
    int    prop_id,       // property identifier  
    string prop_value     // value to be set  
);
```

Parameters

plot_index

[in] Index of [graphical plot](#)

prop_id

[in] The value can be one of the values of the [ENUM_PLOT_PROPERTY_STRING](#) enumeration.

prop_value

[in] The value of the property.

Return Value

If successful, returns [true](#), otherwise [false](#).

PlotIndexGetInteger

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the int, color, bool or char type. There are 2 variants of the function.

Call indicating identifier of the property.

```
int PlotIndexGetInteger (
    int plot_index,      // plotting style index
    int prop_id,         // property identifier
);
```

Call indicating the identifier and modifier of the property.

```
int PlotIndexGetInteger (
    int plot_index,      // plotting index
    int prop_id,         // property identifier
    int prop_modifier    // property modifier
)
```

Parameters

plot_index

[in] Index of the [graphical plotting](#)

prop_id

[in] The value can be one of the values of the [ENUM_PLOT_PROPERTY_INTEGER](#) enumeration.

prop_modifier

[in] Modifier of the specified property. Only color index properties require a modifier.

Note

Function is designed to extract the settings of drawing of the appropriate indicator line. The function works in tandem with the function [PlotIndexSetInteger](#) to copy the drawing properties of one line to another.

Example: an indicator that colors candles depending on the day of the week. Colors for each day are set in a programmatically.



```
#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots 1
//---- plot ColorCandles
#property indicator_label1 "ColorCandles"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double      OpenBuffer[];
double      HighBuffer[];
double      LowBuffer[];
double      CloseBuffer[];
double      ColorCandlesColors[];
color       ColorOfDay[6]={CLR_NONE,clrMediumSlateBlue,
                           clrDarkGoldenrod,clrForestGreen,clrBlueViolet,clrRed};

//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,OpenBuffer,INDICATOR_DATA);
SetIndexBuffer(1,HighBuffer,INDICATOR_DATA);
SetIndexBuffer(2,LowBuffer,INDICATOR_DATA);
SetIndexBuffer(3,CloseBuffer,INDICATOR_DATA);
SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- set number of colors in color buffer
```



```

    PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,6);
//--- set colors for color buffer
    for(int i=1;i<6;i++)
        PlotIndexSetInteger(0,PLOT_LINE_COLOR,i,ColorOfDay[i]);
//--- set accuracy
    IndicatorSetInteger(INDICATOR_DIGITS,_Digits);
    printf("We have %u colors of days",PlotIndexGetInteger(0,PLOT_COLOR_INDEXES));
//---
    }
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//---
    int i;
    MqlDateTime t;
//----
    if(prev_calculated==0) i=0;
    else i=prev_calculated-1;
//----
    while(i<rates_total)
    {
        OpenBuffer[i]=open[i];
        HighBuffer[i]=high[i];
        LowBuffer[i]=low[i];
        CloseBuffer[i]=close[i];
        //--- set color for every candle
        TimeToStruct(time[i],t);
        ColorCandlesColors[i]=t.day_of_week;
        //---
        i++;
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+

```


Object Functions

This is the group of functions intended for working with graphic objects relating to any specified chart.

The functions defining the properties of graphical objects, as well as [ObjectCreate\(\)](#) and [ObjectMove\(\)](#) operations for creating and moving objects along the chart are actually used for sending commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. Visual changes in the properties of graphical objects are implemented when handling the queue of the chart events.

Thus, do not expect an immediate visual update of graphical objects after calling these functions. Generally, the graphical objects on the chart are updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use [ChartRedraw\(\)](#) function to forcefully update the graphical objects.

Function	Action
ObjectCreate	Creates an object of the specified type in a specified chart
ObjectName	Returns the name of an object of the corresponding type in the specified chart (specified chart subwindow)
ObjectDelete	Removes the object with the specified name from the specified chart (from the specified chart subwindow)
ObjectsDeleteAll	Removes all objects of the specified type from the specified chart (from the specified chart subwindow)
ObjectFind	Searches for an object with the specified ID by the name
ObjectGetTimeByValue	Returns the time value for the specified object price value
ObjectGetValueByTime	Returns the price value of an object for the specified time
ObjectMove	Changes the coordinates of the specified object anchor point
ObjectsTotal	Returns the number of objects of the specified type in the specified chart (specified chart subwindow)
ObjectGetDouble	Returns the double value of the corresponding object property
ObjectGetInteger	Returns the integer value of the corresponding object property

<u>ObjectGetString</u>	Returns the string value of the corresponding object property
<u>ObjectSetDouble</u>	Sets the value of the corresponding object property
<u>ObjectSetInteger</u>	Sets the value of the corresponding object property
<u>ObjectSetString</u>	Sets the value of the corresponding object property
<u>TextSetFont</u>	Sets the font for displaying the text using drawing methods (Arial 20 used by default)
<u>TextOut</u>	Transfers the text to the custom array (buffer) designed for creation of a graphical <u>resource</u>
<u>TextGetSize</u>	Returns the string's width and height at the current <u>font settings</u>

Every graphical object should have a name unique within one [chart](#), including its subwindows. Changing of a name of a graphic object generates two events: event of deletion of an object with the old name, and event of creation of an object with a new name.

After an object is created or an [object property](#) is modified it is recommended to call the [ChartRedraw\(\)](#) function, which commands the client terminal to forcibly draw a chart (and all [visible](#) objects in it).

ObjectCreate

The function creates an object with the specified name, type, and the initial coordinates in the specified chart subwindow. During creation up to 30 coordinates can be specified.

```
bool  ObjectCreate(
    long      chart_id,      // chart identifier
    string     name,         // object name
    ENUM_OBJECT type,        // object type
    sub_window nwin,         // window index
    datetime   time1,        // time of the first anchor point
    double     price1,       // price of the first anchor point
    ...
    datetime   timeN=0,      // time of the N-th anchor point
    double     priceN=0,     // price of the N-th anchor point
    ...
    datetime   time30=0,     // time of the 30th anchor point
    double     price30=0     // price of the 30th anchor point
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object. The name must be unique within a chart, including its subwindows.

type

[in] Object type. The value can be one of the values of the [ENUM_OBJECT](#) enumeration.

sub_window

[in] Number of the chart subwindow. 0 means the main chart window. The specified subwindow must exist, otherwise the function returns false.

time1

[in] The time coordinate of the first anchor.

price1

[in] The price coordinate of the first anchor point.

timeN=0

[in] The time coordinate of the N-th anchor point.

priceN=0

[in] The price coordinate of the N-th anchor point.

time30=0

[in] The time coordinate of the thirtieth anchor point.

price30=0

[in] The price coordinate of the thirtieth anchor point.

Return Value

Returns true or false depending on whether the object is created or not. To read more about the [error](#) call [GetLastError\(\)](#). If the object has been created already, the function tries to change its coordinates.

Note

An object name should not exceed 63 characters.

The numbering of the chart subwindows (if there are subwindows with indicators in the chart) starts with 1. The main chart window of the chart is and always has index 0.

The large number of anchor points (up to 30) is implemented for future use. At the same time, the limit of 30 possible anchor points for graphical objects is determined by the limit on the number of parameters (not more than 64) that can be used when calling a function.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

There is a certain number of anchor points that must be specified when creating each [object type](#):

ID	Description	Anchor Points
OBJ_VLINE	Vertical Line	One anchor point. Actually only the time coordinate is used.
OBJ_HLINE	Horizontal Line	One anchor point. Actually only the price coordinate is used.
OBJ_TREND	Trend Line	Two anchor points.
OBJ_TRENDBYANGLE	Trend Line By Angle	Two anchor points.
OBJ_CYCLES	Cycle Lines	Two anchor points.
OBJ_ARROWED_LINE	Arrowed Line	Two anchor points.
OBJ_CHANNEL	Equidistant Channel	Three anchor points.
OBJ_STDDEVCHANNEL	Standard Deviation Channel	Two anchor points.
OBJ_REGRESSION	Linear Regression Channel	Two anchor points.
OBJ_PITCHFORK	Andrews' Pitchfork	Three anchor points.
OBJ_GANNLINE	Gann Line	Two anchor points.
OBJ_GANNFAN	Gann Fan	Two anchor points.
OBJ_GANNGRID	Gann Grid	Two anchor points.
OBJ_FIBO	Fibonacci Retracement	Two anchor points.
OBJ_FIBOTIMES	Fibonacci Time Zones	Two anchor points.
OBJ_FIBOFAN	Fibonacci Fan	Two anchor points.

<u>OBJ_FIBOARC</u>	Fibonacci Arcs	Two anchor points.
<u>OBJ_FIBOCHANNEL</u>	Fibonacci Channel	Three anchor points.
<u>OBJ_EXPANSION</u>	Fibonacci Expansion	Three anchor points.
<u>OBJ_ELLIOTWAVE5</u>	Elliott Motive Wave	Five anchor points.
<u>OBJ_ELLIOTWAVE3</u>	Elliott Correction Wave	Three anchor points.
<u>OBJ_RECTANGLE</u>	Rectangle	Two anchor points.
<u>OBJ_TRIANGLE</u>	Triangle	Three anchor points.
<u>OBJ_ELLIPSE</u>	Ellipse	Three anchor points.
<u>OBJ_ARROW_THUMB_UP</u>	Thumbs Up	One anchor point.
<u>OBJ_ARROW_THUMB_DOWN</u>	Thumbs Down	One anchor point.
<u>OBJ_ARROW_UP</u>	Arrow Up	One anchor point.
<u>OBJ_ARROW_DOWN</u>	Arrow Down	One anchor point.
<u>OBJ_ARROW_STOP</u>	Stop Sign	One anchor point.
<u>OBJ_ARROW_CHECK</u>	Check Sign	One anchor point.
<u>OBJ_ARROW_LEFT_PRICE</u>	Left Price Label	One anchor point.
<u>OBJ_ARROW_RIGHT_PRICE</u>	Right Price Label	One anchor point.
<u>OBJ_ARROW_BUY</u>	Buy Sign	One anchor point.
<u>OBJ_ARROW_SELL</u>	Sell Sign	One anchor point.
<u>OBJ_ARROW</u>	Arrow	One anchor point.
<u>OBJ_TEXT</u>	Text	One anchor point.
<u>OBJ_LABEL</u>	Label	Position is set using the <u>OBJPROP_XDISTANCE</u> and <u>OBJPROP_YDISTANCE</u> properties.
<u>OBJ_BUTTON</u>	Button	Position is set using the <u>OBJPROP_XDISTANCE</u> and <u>OBJPROP_YDISTANCE</u> properties.
<u>OBJ_CHART</u>	Chart	Position is set using the <u>OBJPROP_XDISTANCE</u> and <u>OBJPROP_YDISTANCE</u> properties.
<u>OBJ_BITMAP</u>	Bitmap	One anchor point.
<u>OBJ_BITMAP_LABEL</u>	Bitmap Label	Position is set using the <u>OBJPROP_XDISTANCE</u> and

		<u>OBJPROP_YDISTANCE</u> properties.
<u>OBJ_EDIT</u>	Edit	Position is set using the <u>OBJPROP_XDISTANCE</u> and <u>OBJPROP_YDISTANCE</u> properties.
<u>OBJ_EVENT</u>	The "Event" object corresponding to an event in the economic calendar	One anchor point. Actually only the time coordinate is used.
<u>OBJ_RECTANGLE_LABEL</u>	The "Rectangle label" object for creating and designing the custom graphical interface.	Position is set using the <u>OBJPROP_XDISTANCE</u> and <u>OBJPROP_YDISTANCE</u> properties.

ObjectName

The function returns the name of the corresponding object in the specified chart, in the specified subwindow, of the specified type.

```
string ObjectName(  
    long  chart_id,           // chart identifier  
    int   pos,               // number in the list of objects  
    int   sub_window=-1,     // window index  
    int   type=-1            // object type  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

pos

[in] Ordinal number of the object according to the specified filter by the number and type of the subwindow.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the [ENUM_OBJECT](#) enumeration. -1 means all types.

Return Value

Name of the object is returned in case of success.

Note

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectDelete

The function removes the object with the specified name from the specified chart.

```
bool ObjectDelete(  
    long    chart_id,    // chart identifier  
    string  name         // object name  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of object to be deleted.

Return Value

Returns true if the removal was successful, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Note

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectsDeleteAll

Removes all objects from the specified chart, specified chart subwindow, of the specified type.

```
int ObjectsDeleteAll(  
    long   chart_id,           // chart identifier  
    int    sub_window=-1,     // window index  
    int    type=-1            // object type  
);
```

Removes all objects of the specified type using prefix in object names.

```
int ObjectsDeleteAll(  
    long      chart_id,       // chart ID  
    const string prefix,     // prefix in object name  
    int       sub_window=-1,  // window index  
    int       object_type=-1  // object type  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

prefix

[in] Prefix in object names. All objects whose names start with this set of characters will be removed from chart. You can specify prefix as 'name' or 'name*' - both variants will work the same. If an empty string is specified as the prefix, objects with all possible names will be removed.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the [ENUM_OBJECT](#) enumeration. -1 means all types.

Return Value

Returns the number of deleted objects. To read more about the [error](#) call [GetLastError\(\)](#).

ObjectFind

The function searches for an object with the specified name in the chart with the specified ID.

```
int ObjectFind(  
    long    chart_id,    // chart identifier  
    string  name         // object name  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] The name of the searched object.

Return Value

If successful the function returns the number of the subwindow (0 means the main window of the chart), in which the object is found. If the object is not found, the function returns a negative number. To read more about the [error](#) call [GetLastError\(\)](#).

Note

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectGetTimeByValue

The function returns the time value for the specified price value of the specified object.

```
datetime ObjectGetTimeByValue(  
    long    chart_id,      // chart identifier  
    string  name,          // object name  
    double  value,         // Price  
    int     line_id        // Line number  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

value

[in] Price value.

line_id

[in] Line identifier.

Return Value

The time value for the specified price value of the specified object.

Note

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLIN)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines
- Arrowed line (OBJ_ARROWED_LINE)

See also

[Object Types](#)

ObjectGetValueByTime

The function returns the price value for the specified time value of the specified object.

```
double ObjectGetValueByTime(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    datetime  time,         // Time  
    int       line_id        // Line number  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

time

[in] Time value.

line_id

[in] Line ID.

Return Value

The price value for the specified time value of the specified object.

Note

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLIN)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines
- Arrowed line (OBJ_ARROWED_LINE)

See also

[Object Types](#)

ObjectMove

The function changes coordinates of the specified anchor point of the object.

```
bool ObjectMove(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int        point_index,   // anchor point number  
    datetime   time,         // Time  
    double     price         // Price  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

point_index

[in] Index of the anchor point. The number of anchor points depends on the type of object.

time

[in] Time coordinate of the selected anchor point.

price

[in] Price coordinate of the selected anchor point.

Return Value

If successful, returns true, in case of failure returns false. To read more about the [error](#) call [GetLastError\(\)](#).

ObjectsTotal

The function returns the number of objects in the specified chart, specified subwindow, of the specified type.

```
int ObjectsTotal(  
    long  chart_id,           // chart identifier  
    int   sub_window=-1,     // window index  
    int   type=-1            // object type  
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the [ENUM_OBJECT](#) enumeration. -1 means all types.

Return Value

The number of objects.

ObjectSetDouble

The function sets the value of the corresponding object property. The object property must be of the [double](#) type. There are 2 variants of the function.

Setting property value, without modifier

```
bool ObjectSetDouble(
    long          chart_id,      // chart identifier
    string        name,          // object name
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // property
    double        prop_value     // value
);
```

Setting a property value indicating the modifier

```
bool ObjectSetDouble(
    long          chart_id,      // chart identifier
    string        name,          // object name
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // property
    int          prop_modifier, // modifier
    double        prop_value     // value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_DOUBLE](#) enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Example of creating a Fibonacci object and adding a new level in it

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



```

void OnStart()
{
//--- auxiliary arrays
double high[],low[],price1,price2;
datetime time[],time1,time2;
//--- Copy the open prices - 100 latest bars are enough
int copied=CopyHigh(Symbol(),0,0,100,high);
if(copied<=0)
{
    Print("Failed to copy the values of the High price series");
    return;
}
//--- Copy the close price - 100 latest bars are enough
copied=CopyLow(Symbol(),0,0,100,low);
if(copied<=0)
{
    Print("Failed to copy the values of the Low price series");
    return;
}
//--- Copy the open time for the last 100 bars
copied=CopyTime(Symbol(),0,0,100,time);
if(copied<=0)
{
    Print("Failed to copy the values of the price series of Time");
    return;
}
//--- Organize access to the copied data as to timeseries - backwards
ArraySetAsSeries(high,true);
ArraySetAsSeries(low,true);
ArraySetAsSeries(time,true);

//--- Coordinates of the first anchor point of the Fibo object
price1=high[70];
time1=time[70];
//--- Coordinates of the second anchor point of the Fibo object
price2=low[50];
time2=time[50];

//--- Time to create the Fibo object
bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2);
if(created) // If the object is created successfully
{
    //--- set the color of Fibo levels
    ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
    //--- by the way, how much Fibo levels do we have?
    int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
    Print("Fibo levels before = ",levels);
    //---output to the Journal => number of level:value level_description
    for(int i=0;i<levels;i++)

```



```

    {
        Print(i,": ",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
              " ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
    }
    //--- Try to increase the number of levels per unit
    bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
    if(!modified) // failed to change the number of levels
    {
        Print("Failed to change the number of levels of Fibo, error ",GetLastError())
    }
    //--- just inform
    Print("Fibo levels after = ",ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS));
    //--- set a value for a newly created level
    bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
    if(added) // managed to set a value for the level
    {
        Print("Successfully set one more Fibo level");
        //--- Also do not forget to set the level description
        ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
        ChartRedraw(0);
        //--- Get the actual value of the number of levels in the Fibo object
        levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
        Print("Fibo levels after adding = ",levels);
        //--- once again output all levels - just to make sure
        for(int i=0;i<levels;i++)
        {
            Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                  " ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
        }
    }
    else // Fails if you try to increase the number of levels in the Fibo object
    {
        Print("Failed to set one more Fibo level. Error ",GetLastError());
    }
}
}

```

See also

[Object Types](#), [Object Properties](#)

ObjectSetInteger

The function sets the value of the corresponding object property. The object property must be of the [datetime](#), [int](#), [color](#), [bool](#) or [char](#) type. There are 2 variants of the function.

Setting property value, without modifier

```
bool ObjectSetInteger(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // property
    long          prop_value    // value
);
```

Setting a property value indicating the modifier

```
bool ObjectSetInteger(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // property
    int           prop_modifier, // modifier
    long          prop_value    // value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_INTEGER](#) enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the [error](#) call [GetLastError\(\)](#).

An example of how to create a table of [Web colors](#)

```
//+-----+
//|                                     Table of Web Colors|
```



```

//|                                     Copyright 2011, MetaQuotes Software Corp |
//|                                     https://www.metaquotes.net |
//+-----+
#define X_SIZE 140          // width of an edit object
#define Y_SIZE 33          // height of an edit object
//+-----+
//| Array of web colors |
//+-----+
color ExtClr[140]=
{
    clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,clrBisque,clr
    clrBlue,clrBlueViolet,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,clrChocolate
    clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod,clrDarkGray
    clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed,clrDarkSalmon
    clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDeepPink,clrDeep
    clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clrGainsboro,clr
    clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clrIndianRed,clr
    clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,clrLightCoral
    clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmon,clrLightSea
    clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen,clrLinen,clr
    clrMediumAquamarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMediumSeaGreen
    clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightBlue,clrMintCh
    clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOrangeRed,clrOr
    clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPuff,clrPeru,clr
    clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clrSandyBrown,clr
    clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpringGreen,clr
    clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteSmoke,clrYellow
};
//+-----+
//| Creating and initializing an edit object |
//+-----+
void CreateColorBox(int x,int y,color c)
{
    //--- generate a name for a new edit object
    string name="ColorBox_"+(string)x+"_"+(string)y;
    //--- create a new edit object
    if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
    {
        Print("Cannot create: ",name,"");
        return;
    }
    //--- set coordinates, width and height
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x*X_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y*Y_SIZE);
    ObjectSetInteger(0,name,OBJPROP_XSIZE,X_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YSIZE,Y_SIZE);
    //--- set text color
    if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
    else           ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
}

```



```
//--- set background color
    ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- set text
    ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create 7x20 table of colored edit objects
    for(uint i=0;i<140;i++)
        CreateColorBox(i%7,i/7,ExtClr[i]);
}
```

See also

[Object Types](#), [Object Properties](#)

ObjectSetString

The function sets the value of the corresponding object property. The object property must be of the [string](#) type. There are 2 variants of the function.

Setting property value, without modifier

```
bool ObjectSetString(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_STRING prop_id, // property
    string        prop_value    // value
);
```

Setting a property value indicating the modifier

```
bool ObjectSetString(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_STRING prop_id, // property
    int          prop_modifier, // modifier
    string        prop_value    // value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_STRING](#) enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Note

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectGetDouble

The function returns the value of the corresponding object property. The object property must be of the [double](#) type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double ObjectGetDouble(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // property identifier
    int          prop_modifier=0 // property modifier, if required
);
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool ObjectGetDouble(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // property identifier
    int          prop_modifier, // property modifier
    double&       double_var    // here we accept the property value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_DOUBLE](#) enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

double_var

[out] Variable of the double type that received the value of the requested property.

Return Value

Value of the double type for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the double_var variable, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

ObjectGetInteger

The function returns the value of the corresponding object property. The object property must be of the [datetime](#), [int](#), [color](#), [bool](#) or [char](#) type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long ObjectGetInteger(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // property identifier
    int          prop_modifier=0 // property modifier, if required
);
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool ObjectGetInteger(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // property identifier
    int          prop_modifier, // property modifier
    long&         long_var       // here we accept the property value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_INTEGER](#) enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

long_var

[out] Variable of the long type that receives the value of the requested property.

Return Value

The long value for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the long_var variable, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

ObjectGetString

The function returns the value of the corresponding object property. The object property must be of the [string](#) type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string ObjectGetString(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_STRING prop_id, // property identifier
    int          prop_modifier=0 // property modifier, if required
);
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool ObjectGetString(
    long          chart_id,      // chart identifier
    string        name,         // object name
    ENUM_OBJECT_PROPERTY_STRING prop_id, // property identifier
    int          prop_modifier, // property modifier
    string&       string_var    // here we accept the property value
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the [ENUM_OBJECT_PROPERTY_STRING](#) enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in [Fibonacci tools](#) and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

string_var

[out] Variable of the string type that receives the value of the requested properties.

Return Value

String value for the first version of the call.

For the second version of the call returns true, if this property is maintained and the value has been placed into the *string_var* variable, otherwise returns false. To read more about the [error](#) call [GetLastError\(\)](#).

Note

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the [OnChartEvent\(\)](#) function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

TextSetFont

The function sets the font for displaying the text using drawing methods and returns the result of that operation. Arial font with the size -120 (12 pt) is used by default.

```
bool TextSetFont(  
    const string name,           // font name or path to font file on the disk  
    int size,                   // font size  
    uint flags,                 // combination of flags  
    int orientation=0           // text slope angle  
);
```

Parameters

name

[in] Font name in the system or the name of the resource containing the font or the path to font file on the disk.

size

[in] The font size that can be set using positive and negative values. In case of positive values, the size of a displayed text does not depend on the operating system's font size settings. In case of negative values, the value is set in tenths of a point and the text size depends on the operating system settings ("standard scale" or "large scale"). See the Note below for more information about the differences between the modes.

flags

[in] Combination of [flags](#) describing font style.

orientation

[in] Text's horizontal inclination to X axis, the unit of measurement is 0.1 degrees. It means that *orientation=450* stands for inclination equal to 45 degrees.

Return Value

Returns true if the current font is successfully installed, otherwise false. Possible code errors:

- `ERR_INVALID_PARAMETER(4003)` - *name* presents NULL or "" (empty string),
- `ERR_INTERNAL_ERROR(4001)` - operating system error (for example, an attempt to create a non-existent font).

Note

If ":::" is used in font name, the font is downloaded from [EX5 resource](#). If *name* font name is specified with an extension, the font is downloaded from the file, if the path starts from "\" or "/", the file is searched relative to MQL5 directory. Otherwise, it is searched relative to the path of EX5 file which called `TextSetFont()` function.

The font size is set using positive or negative values. This fact defines the dependence of the text size from the operating system settings (size scale).

- If the size is specified using a positive number, this size is transformed into physical measurement units of a device (pixels) when changing a logical font into a physical one, and this size corresponds to the height of the symbol glyphs picked from the available fonts. This case is not recommended when the texts displayed by [TextOut\(\)](#) function and the ones displayed by [OBJ_LABEL](#) ("Label") graphical object are to be used together on the chart.

- If the size is specified using a negative number, this number is supposed to be set in tenths of a logical point and is divided by 10 (for example, -350 is equal to 35 logical points). An obtained value is then transformed into physical measurement units of a device (pixels) and corresponds to the absolute value of the height of a symbol picked from the available fonts. Multiply the font size specified in the object properties by -10 to make the size of a text on the screen similar to the one in [OBJ_LABEL](#) object.

The flags can be used as the combination of style flags with one of the flags specifying the font width. Flag names are shown below.

Flags for specifying font style

Flag	Description
FONT_ITALIC	Italic
FONT_UNDERLINE	Underline
FONT_STRIKEOUT	Strikeout

Flags for specifying font width

Flag
FW_DONTCARE
FW_THIN
FW_EXTRALIGHT
FW_ULTRALIGHT
FW_LIGHT
FW_NORMAL
FW_REGULAR
FW_MEDIUM
FW_SEMIBOLD
FW_DEMIBOLD
FW_BOLD
FW_EXTRABOLD
FW_ULTRABOLD
FW_HEAVY
FW_BLACK

See also

[Resources](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextOut\(\)](#)

TextOut

The function displays a text in a custom array (buffer) and returns the result of that operation. The array is designed to create the graphical [resource](#).

```
bool TextOut(  
    const string      text,           // displayed text  
    int              x,               // X coordinate  
    int              y,               // Y coordinate  
    uint             anchor,          // anchor type  
    uint             &data[],         // output buffer  
    uint             width,           // buffer width in pixels  
    uint             height,          // buffer height in pixels  
    uint             color,           // text color  
    ENUM_COLOR_FORMAT color_format    // color format for output  
);
```

Parameters

text

[in] Displayed text that will be written to the buffer. Only one-lined text is displayed.

x

[in] X coordinate of the anchor point of the displayed text.

y

[in] Y coordinate of the anchor point of the displayed text.

anchor

[in] The value out of the 9 pre-defined methods of the displayed text's anchor point location. The value is set by a combination of two flags - flags of horizontal and vertical text align. Flag names are listed in the Note below.

data[]

[in] Buffer, in which text is displayed. The buffer is used to create the graphical [resource](#).

width

[in] Buffer width in pixels.

height

[in] Buffer height in pixels.

color

[in] Text color.

color_format

[in] Color format is set by [ENUM_COLOR_FORMAT](#) enumeration value.

Return Value

Returns true if successful, otherwise false.

Note

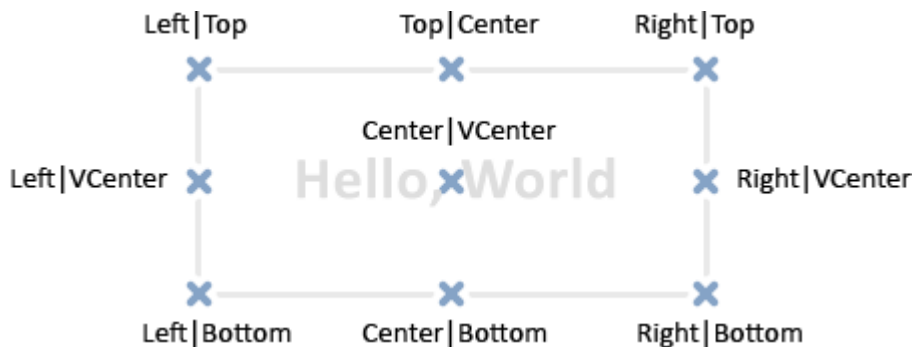
Anchor point specified by *anchor* is a combination of two flags of horizontal and vertical text align.
Horizontal text align flags:

- TA_LEFT - anchor point on the left side of the bounding box
- TA_CENTER - horizontal anchor point is located at the center of the bounding box
- TA_RIGHT - anchor point on the right side of the bounding box

Vertical text align flags:

- TA_TOP - anchor point at the upper side of the bounding box
- TA_VCENTER - vertical anchor point is located at the center of the bounding box
- TA_BOTTOM - anchor point at the lower side of the bounding box

Possible combinations of flags and specified anchor points are shown in the image.



Example:

```
//--- width and height of the canvas (used for drawing)
#define IMG_WIDTH 200
#define IMG_HEIGHT 200
//--- display the parameter window before launching the script
#property script_show_inputs
//--- enable to set color format
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- drawing array (buffer)
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create OBJ_BITMAP_LABEL object for drawing
ObjectCreate(0,"CLOCK",OBJ_BITMAP_LABEL,0,0,0);
//--- specify the name of the graphical resource for writing in CLOCK object
ObjectSetString(0,"CLOCK",OBJPROP_BMPFILE,"::IMG");

//--- auxiliary variables
double a;           // arrow corner
uint nm=2700;       // minute corner
uint nh=2700*12;    // hour counter
uint w,h;           // variables for receiving text string sizes
```



```

    int    x,y;           // variables for calculation of the current coordinates of text

//--- rotate clock hands in an infinite loop, till the script is stopped
while(!IsStopped())
{
    //--- clear the clock drawing buffer array
    ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
    //--- set the font for drawing digits for the clock-face
    TextSetFont("Arial",-200,FW_EXTRABOLD,0);
    //--- draw the clock-face
    for(int i=1;i<=12;i++)
    {
        //--- receive the size of the current hour on the clock-face
        TextGetSize(string(i),w,h);
        //--- calculate the coordinates of the current hour on the clock-face
        a=-((i*300)%3600*M_PI)/1800.0;
        x=IMG_WIDTH/2-int(sin(a)*80+0.5+w/2);
        y=IMG_HEIGHT/2-int(cos(a)*80+0.5+h/2);
        //--- output the hour on the clock-face to ExtImg[] buffer
        TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_1);
    }
    //--- now, specify the font for drawing the minute hand
    TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nm%3600));
    //--- receive the size of the minute hand
    TextGetSize("----->",w,h);
    //--- calculate the coordinates of the minute hand on the clock-face
    a=-(nm%3600*M_PI)/1800.0;
    x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
    y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
    //--- output of the minute hand to the clock-face in ExtImg[]buffer
    TextOut("----->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_1);

    //--- now, set the font for drawing the minute hand
    TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nh/12%3600));
    TextGetSize("==>",w,h);
    //--- calculate the coordinates of the hour hand on the clock-face
    a=-(nh/12%3600*M_PI)/1800.0;
    x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
    y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
    //--- output of the hour hand on the clock-face to ExtImg[] buffer
    TextOut("==>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_form);

    //--- update the graphical resource
    ResourceCreate("::IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr_format);
    //--- forced chart update
    ChartRedraw();

    //--- increase hour and minute counters
    nm+=60;
    nh+=60;
}

```



```
    //--- keeping a short pause between the frames
    Sleep(10);
}
//--- delete CLOCK object when completing the script's operation
ObjectDelete(0,"CLOCK");
//---
}
```

See also

[Resources](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextGetSize\(\)](#), [TextSetFont\(\)](#)

TextGetSize

The function returns the line width and height at the current [font settings](#).

```
bool TextGetSize(  
    const string      text,           // text string  
    uint&             width,          // buffer width in pixels  
    uint&             height         // buffer height in pixels  
);
```

Parameters

text

[in] String, for which length and width should be obtained.

width

[out] Input parameter for receiving width.

height

[out] Input parameter for receiving height.

Return Value

Returns true if successful, otherwise false. Possible code errors:

- `ERR_INTERNAL_ERROR(4001)` - operating system error.

See also

[Resources](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextSetFont\(\)](#), [TextOut\(\)](#)

Technical Indicator Functions

All functions like `iMA`, `iAC`, `iMACD`, `ilchimoku` etc. create a copy of the corresponding technical indicator in the global cache of the client terminal. If a copy of the indicator with such parameters already exists, the new copy is not created, and the counter of references to the existing copy increases.

These functions return the handle of the appropriate copy of the indicator. Further, using this handle, you can receive data calculated by the corresponding indicator. The corresponding buffer data (technical indicators contain calculated data in their internal buffers, which can vary from 1 to 5, depending on the indicator) can be copied to a mql5-program using the [CopyBuffer\(\)](#) function.

You can't refer to the indicator data right after it has been created, because calculation of indicator values requires some time, so it's better to create indicator handles in `OnInit()`. Function [iCustom\(\)](#) creates the corresponding custom indicator, and returns its handle in case it is successfully create. Custom indicators can contain up to 512 indicator buffers, the contents of which can also be obtained by the [CopyBuffer\(\)](#) function, using the obtained handle.

There is a universal method for creating any technical indicator using the [IndicatorCreate\(\)](#) function. This function accepts the following data as input parameters:

- symbol name;
- timeframe;
- type of the indicator to create;
- number of input parameters of the indicator;
- an array of [MqlParam](#) type containing all the necessary input parameters.

The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note. Repeated call of the indicator function with the same parameters within one mql5-program does not lead to a multiple increase of the reference counter; the counter will be increased only once by 1. However, it's recommended to get the indicators handles in function [OnInit\(\)](#) or in the class constructor, and further use these handles in other functions. The reference counter decreases when a mql5-program is deinitialized.

All indicator functions have at least 2 parameters - symbol and period. The [NULL](#) value of the symbol means the current symbol, the 0 value of the period means the current [timeframe](#).

Function	Returns the handle of the indicator:
iAC	Accelerator Oscillator
iAD	Accumulation/Distribution
iADX	Average Directional Index
iADXWilder	Average Directional Index by Welles Wilder
iAlligator	Alligator
iAMA	Adaptive Moving Average

<u>iAO</u>	Awesome Oscillator
<u>iATR</u>	Average True Range
<u>iBearsPower</u>	Bears Power
<u>iBands</u>	Bollinger Bands®
<u>iBullsPower</u>	Bulls Power
<u>iCCI</u>	Commodity Channel Index
<u>iChaikin</u>	Chaikin Oscillator
<u>iCustom</u>	Custom indicator
<u>iDEMA</u>	Double Exponential Moving Average
<u>iDeMarker</u>	DeMarker
<u>iEnvelopes</u>	Envelopes
<u>iForce</u>	Force Index
<u>iFractals</u>	Fractals
<u>iFrAMA</u>	Fractal Adaptive Moving Average
<u>iGator</u>	Gator Oscillator
<u>ilchimoku</u>	Ichimoku Kinko Hyo
<u>iBWMFI</u>	Market Facilitation Index by Bill Williams
<u>iMomentum</u>	Momentum
<u>iMFI</u>	Money Flow Index
<u>iMA</u>	Moving Average
<u>iOsMA</u>	Moving Average of Oscillator (MACD histogram)
<u>iMACD</u>	Moving Averages Convergence-Divergence
<u>iOBV</u>	On Balance Volume
<u>iSAR</u>	Parabolic Stop And Reverse System
<u>iRSI</u>	Relative Strength Index
<u>iRVI</u>	Relative Vigor Index
<u>iStdDev</u>	Standard Deviation
<u>iStochastic</u>	Stochastic Oscillator
<u>iTEMA</u>	Triple Exponential Moving Average
<u>iTriX</u>	Triple Exponential Moving Averages Oscillator
<u>iWPR</u>	Williams' Percent Range

<u>iVIDyA</u>	Variable Index Dynamic Average
<u>iVolumes</u>	Volumes

iAC

The function creates Accelerator Oscillator in a global cache of the client terminal and returns its handle. It has only one buffer.

```
int iAC(
    string          symbol,      // symbol name
    ENUM_TIMEFRAMES period      // period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) enumeration values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iAC.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAC technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plotting of iAC
#property indicator_label1 "iAC"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
#property indicator_color1 clrGreen, clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
```



```

//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iAC,           // use iAC
    Call_IndicatorCreate // use IndicatorCreate
};

//--- input parameters
input Creation      type=Call_iAC;           // type of the function
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe

//--- indicator buffers
double iACBuffer[];
double iACColors[];

//--- variable for storing the handle of the iAC indicator
int    handle;

//--- variable for storing
string name=symbol;

//--- name of the indicator on a chart
string short_name;

//--- we will keep the number of values in the Accelerator Oscillator indicator
int    bars_calculated=0;

//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,iACBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iACColors,INDICATOR_COLOR_INDEX);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iAC)
        handle=iAC(name,period);
    else
        handle=IndicatorCreate(name,period,IND_AC);
    //--- if the handle is not created
    if(handle==INVALID_HANDLE)

```



```

{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iAC indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());

    //--- the indicator is stopped early
    return(INIT_FAILED);
}

//--- show the symbol/timeframe the Accelerator Oscillator indicator is calculated for
short_name=StringFormat("iAC(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])

{
    //--- number of values copied from the iAC indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of va
    //---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iACBuffer array is greater than the number of values in the iAC ind
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {

```



```

        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

    //--- fill the iACBuffer and iACColors arrays with values from the Accelerator Oscillator
    //--- if FillArraysFromBuffer returns false, it means the information is not ready yet
    if(!FillArraysFromBuffer(iACBuffer,iACColors,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Accelerator Oscillator indicator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iAC indicator |
//+-----+

bool FillArraysFromBuffer(double &values[], // indicator buffer of Accelerator
                          double &color_indexes[], // color buffer (for storing of color indexes)
                          int ind_handle, // handle of the iAC indicator
                          int amount // number of copied values
                          )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iACBuffer array with values from the indicator buffer that has been calculated
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAC indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- now copy the indexes of colors
    if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy color values from the iAC indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}

```



```
//+-----+  
//| Indicator deinitialization function |  
//+-----+  
void OnDeinit(const int reason)  
{  
    //--- clear the chart after deleting the indicator  
    Comment("");  
}
```


iAD

The function returns the handle of the Accumulation/Distribution indicator. It has only one buffer.

```
int iAD(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) enumeration values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be any of [ENUM_APPLIED_VOLUME](#) values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iAD.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAD
#property indicator_label1   "iAD"
#property indicator_type1    DRAW_LINE
#property indicator_color1   clrLightSeaGreen
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iAD,           // use iAD
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iAD;           // type of the function
input ENUM_APPLIED_VOLUME volumes;           // volume used
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iADBuffer[];
//--- variable for storing the handle of the iAD indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Accumulation/Distribution indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iADBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iAD)
        handle=iAD(name,period,volumes);
    else
    {
        //--- fill the structure with parameters of the indicator

```



```

    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=volumes;
    handle=IndicatorCreate(name,period,IND_AD,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iAD indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Accumulation/Distribution indicator is calculated
short_name=StringFormat("iAD(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iAD indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {

```



```

    //--- if the iADBuffer array is greater than the number of values in the iAD inc
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
    }
else
{
    //--- it means that it's not the first time of the indicator calculation, and s
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the iADBuffer array with values of the Accumulation/Distribution indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
if(!FillArrayFromBuffer(iADBuffer,handle,values_to_copy)) return(0);
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                          TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                          short_name,
                          values_to_copy);
//--- display the service message on the chart
Comment(comm);
//--- memorize the number of values in the Accumulation/Distribution indicator
bars_calculated=calculated;
//--- return the prev_calculated value for the next call
return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iAD indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of the Accumulation/D
                        int ind_handle, // handle of the iAD indicator
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iADBuffer array with values from the indicator buffer that ha
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAD indicator, error code %d",GetLastE
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |

```



```
//+-----+  
void OnDeinit(const int reason)  
{  
    ///--- clear the chart after deleting the indicator  
    Comment("");  
}
```


iADX

The function returns the handle of the Average Directional Movement Index indicator.

```
int iADX(
    string      symbol,          // symbol name
    ENUM_TIMEFRAMES period,      // period
    int         adx_period       // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

adx_period

[in] Period to calculate the index.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Example:

```
//+-----+
//|                                     Demo_iADX.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```



```

//--- plot ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iADX,          // use iADX
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iADX;          // type of the function
input int           adx_period=14;           // period of calculation
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double      ADXBuffer[];
double      DI_plusBuffer[];
double      DI_minusBuffer[];
//--- variable for storing the handle of the iADX indicator
int  handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average Directional Movement Index indicator
int  bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers

```



```

SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
name=symbol;
//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iADX)
    handle=iADX(name,period,adx_period);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=adx_period;
    handle=IndicatorCreate(name,period,IND_ADX,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iADX indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Average Directional Movement Index indicator is calculated for
short_name=StringFormat("iADX(%s/%s period=%d)",name,EnumToString(period),adx_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],

```



```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//--- number of values copied from the iADX indicator
        int values_to_copy;
//--- determine the number of values calculated in the indicator
        int calculated=BarsCalculated(handle);
        if(calculated<=0)
        {
            PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
            return(0);
        }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to recalculate the indicator for two or more bars (it means that the indicator is not calculated for a long time)
        if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
        {
            //--- if the iADXBuffer array is greater than the number of values in the iADX indicator
            //--- otherwise, we copy less than the size of indicator buffers
            if(calculated>rates_total) values_to_copy=rates_total;
            else
                values_to_copy=calculated;
        }
        else
        {
            //--- it means that it's not the first time of the indicator calculation, and so we copy only one bar
            //--- for calculation not more than one bar is added
            values_to_copy=(rates_total-prev_calculated)+1;
        }
//--- fill the array with values of the Average Directional Movement Index indicator
//--- if FillArraysFromBuffer returns false, it means the information is not ready yet
        if(!FillArraysFromBuffer(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_copy))
//--- form the message
        string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                   TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                   short_name,
                                   values_to_copy);
//--- display the service message on the chart
        Comment(comm);
//--- memorize the number of values in the Average Directional Movement Index indicator
        bars_calculated=calculated;
//--- return the prev_calculated value for the next call
        return(rates_total);
    }
//+-----+
//| Filling indicator buffers from the iADX indicator |
//+-----+

```



```

bool FillArraysFromBuffers(double &adx_values[],      // indicator buffer of the ADX
                           double &DIplus_values[],  // indicator buffer for DI+
                           double &DIminus_values[], // indicator buffer for DI-
                           int ind_handle,           // handle of the iADX indicator
                           int amount                // number of copied values
                           )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iADXBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not
        return(false);
    }

    //--- fill a part of the DI_plusBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not
        return(false);
    }

    //--- fill a part of the DI_minusBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not
        return(false);
    }

    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iADXWilder

The function returns the handle of Average Directional Movement Index by Welles Wilder.

```
int iADXWilder(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    int         adx_period   // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

adx_period

[in] Period to calculate the index.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Example:

```
//+-----+
//|                                     iADXWilder.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADXWilder technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```



```

//--- plot ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iADXWilder,      // use iADXWilder
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iADXWilder;  // type of the function
input int               adx_period=14;        // period of calculation
input string            symbol=" ";           // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double      ADXBuffer[];
double      DI_plusBuffer[];
double      DI_minusBuffer[];
//--- variable for storing the handle of the iADXWilder indicator
int  handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average Directional Movement Index by V
int  bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers

```



```

SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
name=symbol;
//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iADXWilder)
    handle=iADXWilder(name,period,adx_period);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=adx_period;
    handle=IndicatorCreate(name,period,IND_ADXW,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iADXWilder indicator for the symbol
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Average Directional Movement Index by Welles Wilder
short_name=StringFormat("iADXWilder(%s/%s period=%d)",name,EnumToString(period),adx
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],

```



```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//--- number of values copied from the iADXWilder indicator
        int values_to_copy;
//--- determine the number of values calculated in the indicator
        int calculated=BarsCalculated(handle);
        if(calculated<=0)
        {
            PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
            return(0);
        }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to calculate the indicator for two or more bars (it means that the indicator is not yet calculated)
        if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
        {
            //--- if the iADXBuffer array is greater than the number of values in the iADXWilder indicator
            //--- otherwise, we copy less than the size of indicator buffers
            if(calculated>rates_total) values_to_copy=rates_total;
            else
                values_to_copy=calculated;
        }
        else
        {
            //--- it means that it's not the first time of the indicator calculation, and so we copy only one bar
            //--- for calculation not more than one bar is added
            values_to_copy=(rates_total-prev_calculated)+1;
        }
//--- fill the array with values of the Average Directional Movement Index by Welles Wilder
//--- if FillArraysFromBuffer returns false, it means the information is not ready yet
        if(!FillArraysFromBuffer(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_copy))
//--- form the message
        string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);
//--- display the service message on the chart
        Comment(comm);
//--- memorize the number of values in the Average Directional Movement Index indicator
        bars_calculated=calculated;
//--- return the prev_calculated value for the next call
        return(rates_total);
    }
//+-----+
//| Filling indicator buffers from the iADXWilder indicator |
//+-----+

```



```

bool FillArraysFromBuffers(double &adx_values[],      // indicator buffer of the ADX
                           double &DIplus_values[],  // indicator buffer for DI+
                           double &DIminus_values[], // indicator buffer for DI-
                           int ind_handle,           // handle of the iADXWilder indicator
                           int amount                // number of copied values
                           )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iADXBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not copied
        return(false);
    }

    //--- fill a part of the DI_plusBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not copied
        return(false);
    }

    //--- fill a part of the DI_minusBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not copied
        return(false);
    }

    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iAlligator

The function returns the handle of the Alligator indicator.

```
int iAlligator(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             jaw_period,      // period for the calculation of jaws
    int             jaw_shift,       // horizontal shift of jaws
    int             teeth_period,    // period for the calculation of teeth
    int             teeth_shift,     // horizontal shift of teeth
    int             lips_period,     // period for the calculation of lips
    int             lips_shift,      // horizontal shift of lips
    ENUM_MA_METHOD  ma_method,       // type of smoothing
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

jaw_period

[in] Averaging period for the blue line (Alligator's Jaw)

jaw_shift

[in] The shift of the blue line relative to the price chart.

teeth_period

[in] Averaging period for the red line (Alligator's Teeth).

teeth_shift

[in] The shift of the red line relative to the price chart.

lips_period

[in] Averaging period for the green line (Alligator's lips).

lips_shift

[in] The shift of the green line relative to the price chart.

ma_method

[in] The method of averaging. Can be any of the [ENUM_MA_METHOD](#) values.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - GATORJAW_LINE, 1 - GATORTEETH_LINE, 2 - GATORLIPS_LINE.

Example:

```
//+-----+
//|                                     Demo_iAlligator.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAlligator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Alligator."

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 3
//--- plot Jaws
#property indicator_label1 "Jaws"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot Teeth
#property indicator_label2 "Teeth"
#property indicator_type2  DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot Lips
#property indicator_label3 "Lips"
#property indicator_type3  DRAW_LINE
#property indicator_color3  clrLime
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+-----+
```



```

//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iAlligator,      // use iAlligator
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iAlligator;  // type of the function
input string            symbol=" ";           // symbol
input ENUM_TIMEFRAMES  period=PERIOD_CURRENT; // timeframe
input int               jaw_period=13;        // period of the Jaw line
input int              jaw_shift=8;           // shift of the Jaw line
input int              teeth_period=8;        // period of the Teeth line
input int              teeth_shift=5;         // shift of the Teeth line
input int              lips_period=5;         // period of the Lips line
input int              lips_shift=3;          // shift of the Lips line
input ENUM_MA_METHOD    MA_method=MODE_SMMMA; // method of averaging of the Alligator
input ENUM_APPLIED_PRICE applied_price=PRICE_MEDIAN; // type of price used for calculation
//--- indicator buffers
double      JawsBuffer[];
double      TeethBuffer[];
double      LipsBuffer[];
//--- variable for storing the handle of the iAlligator indicator
int  handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Alligator indicator
int  bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,JawsBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,TeethBuffer,INDICATOR_DATA);
    SetIndexBuffer(2,LipsBuffer,INDICATOR_DATA);
    //--- set shift of each line
    PlotIndexSetInteger(0,PLOT_SHIFT,jaw_shift);
    PlotIndexSetInteger(1,PLOT_SHIFT,teeth_shift);
    PlotIndexSetInteger(2,PLOT_SHIFT,lips_shift);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
}

```



```

//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iAlligator)
    handle=iAlligator(name,period,jaw_period,jaw_shift,teeth_period,
                      teeth_shift,lips_period,lips_shift,MA_method,applied_price);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[8];
    //--- periods and shifts of the Alligator lines
    pars[0].type=TYPE_INT;
    pars[0].integer_value=jaw_period;
    pars[1].type=TYPE_INT;
    pars[1].integer_value=jaw_shift;
    pars[2].type=TYPE_INT;
    pars[2].integer_value=teeth_period;
    pars[3].type=TYPE_INT;
    pars[3].integer_value=teeth_shift;
    pars[4].type=TYPE_INT;
    pars[4].integer_value=lips_period;
    pars[5].type=TYPE_INT;
    pars[5].integer_value=lips_shift;
    //--- type of smoothing
    pars[6].type=TYPE_INT;
    pars[6].integer_value=MA_method;
    //--- type of price
    pars[7].type=TYPE_INT;
    pars[7].integer_value=applied_price;
    //--- create handle
    handle=IndicatorCreate(name,period,IND_ALLIGATOR,8,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iAlligator indicator for the symbol
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Alligator indicator is calculated for
short_name=StringFormat("iAlligator(%s/%s, %d,%d,%d,%d,%d,%d)",name,EnumToString(pe

```



```

        jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,1

IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- number of values copied from the iAlligator indicator
int values_to_copy;
//--- determine the number of values calculated in the indicator
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
    return(0);
}
//--- if it is the first start of calculation of the indicator or if the number of values
//---or if it is necessary to calculate the indicator for two or more bars (it means
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- if the JawsBuffer array is greater than the number of values in the iAlligator
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- fill the arrays with values of the Alligator indicator
//--- if FillArraysFromBuffer returns false, it means the information is not ready yet
if(!FillArraysFromBuffers(JawsBuffer,jaw_shift,TeethBuffer,teeth_shift,LipsBuffer,lips_shift))
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                          TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),

```



```

        short_name,
        values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Alligator indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iAlligator indicator |
//+-----+
bool FillArraysFromBuffers(double &jaws_buffer[], // indicator buffer for the Jaw line
                           int j_shift,          // shift of the Jaw line
                           double &teeth_buffer[], // indicator buffer for the Teeth line
                           int t_shift,          // shift of the Teeth line
                           double &lips_buffer[], // indicator buffer for the Lips line
                           int l_shift,          // shift of the Lips line
                           int ind_handle,        // handle of the iAlligator indicator
                           int amount            // number of copied values
                           )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the JawsBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,0,-j_shift,amount,jaws_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

//--- fill a part of the TeethBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,1,-t_shift,amount,teeth_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

//--- fill a part of the LipsBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,2,-l_shift,amount,lips_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
}

```



```
    }  
    //--- everything is fine  
    return(true);  
}  
//+-----+  
//| Indicator deinitialization function |  
//+-----+  
void OnDeinit(const int reason)  
{  
    //--- clear the chart after deleting the indicator  
    Comment("");  
}
```


iAMA

The function returns the handle of the Adaptive Moving Average indicator. It has only one buffer.

```
int iAMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ama_period,      // average period for AMA
    int             fast_ma_period,  // fast MA period
    int             slow_ma_period,  // slow MA period
    int             ama_shift,       // horizontal shift of the indicator
    ENUM_APPLIED_PRICE applied_price // type of the price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ama_period

[in] The calculation period, on which the efficiency coefficient is calculated.

fast_ma_period

[in] Fast period for the smoothing coefficient calculation for a rapid market.

slow_ma_period

[in] Slow period for the smoothing coefficient calculation in the absence of trend.

ama_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iAMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
```



```

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard AMA."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot iAMA
#property indicator_label1 "iAMA"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1 1

//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iAMA,          // use iAMA
    Call_IndicatorCreate // use IndicatorCreate
};

//--- input parameters
input Creation      type=Call_iAMA;          // type of the function
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
input int            ama_period=15;          // period of calculation
input int            fast_ma_period=2;       // period of fast MA
input int            slow_ma_period=30;      // period of slow MA
input int            ama_shift=0;            // horizontal shift
input ENUM_APPLIED_PRICE applied_price;      // type of price
//--- indicator buffer
double            iAMABuffer[];
//--- variable for storing the handle of the iAMA indicator
int              handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Adaptive Moving Average indicator
int              bars_calculated=0;

//+-----+
//| Custom indicator initialization function |
//+-----+

```



```

int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,iAMABuffer,INDICATOR_DATA);
    //--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ama_shift);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iAMA)
        handle=iAMA(name,period,ama_period,fast_ma_period,slow_ma_period,ama_shift,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[5];
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ama_period;
        pars[1].type=TYPE_INT;
        pars[1].integer_value=fast_ma_period;
        pars[2].type=TYPE_INT;
        pars[2].integer_value=slow_ma_period;
        pars[3].type=TYPE_INT;
        pars[3].integer_value=ama_shift;
        //--- type of price
        pars[4].type=TYPE_INT;
        pars[4].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_AMA,5,pars);
    }
    //--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iAMA indicator for the symbol %s/%s,
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
    //--- show the symbol/timeframe the Adaptive Moving Average indicator is calculated for

```



```

    short_name=StringFormat("iAMA(%s/%s,%d,%d,%d,d)",name,EnumToString(period),ama_peri
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- number of values copied from the iAMA indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iAMABuffer array is greater than the number of values in the iAMA
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the arrays with values of the Adaptive Moving Average indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
    if(!FillArrayFromBuffer(iAMABuffer,ama_shift,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),

```



```

        short_name,
        values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Adaptive Moving Average indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffer from the iAMA indicator |
//+-----+
bool FillArrayFromBuffer(double &ama_buffer[], // indicator buffer of the AMA line
                        int a_shift,          // shift of the AMA line
                        int ind_handle,        // handle of the iAMA indicator
                        int amount            // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iAMABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-a_shift,amount,ama_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iAO

The function returns the handle of the Awesome Oscillator indicator. It has only one buffer.

```
int iAO(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period   // period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iAO.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAO technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- the iAO plot
#property indicator_label1 "iAO"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
#property indicator_color1 clrGreen,clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
```



```

//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iAO,           // use iAO
    Call_IndicatorCreate // use IndicatorCreate
};

//--- input parameters
input Creation      type=Call_iAO;           // type of the function
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe

//--- indicator buffers
double      iAOBuffer[];
double      iAOCOLORS[];
//--- variable for storing the handle of the iAO indicator
int  handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Awesome Oscillator indicator
int  bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,iAOBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iAOCOLORS,INDICATOR_COLOR_INDEX);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iAO)
        handle=iAO(name,period);
    else
        handle=IndicatorCreate(name,period,IND_AO);
    //--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {

```



```

    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iAO indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}

//--- show the symbol/timeframe the Awesome Oscillator indicator is calculated for
short_name=StringFormat("iAO(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iAO indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iAObuffer array is greater than the number of values in the iAO indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s

```



```

        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

    //--- fill the iAOBuffer and iAOCColors arrays with values from the Awesome Oscillator
    //--- if FillArraysFromBuffer returns false, it means the information is not ready yet
    if(!FillArraysFromBuffer(iAOBuffer,iAOCColors,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Awesome Oscillator indicator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iAO indicator |
//+-----+

bool FillArraysFromBuffer(double &values[], // indicator buffer of Awesome Oscillator
                          double &color_indexes[], // color buffer (for storing of color indexes)
                          int ind_handle, // handle of the iAO indicator
                          int amount // number of copied values
                          )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iAOBuffer array with values from the indicator buffer that has been calculated
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iAO indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- now copy the indexes of colors
    if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy color values from the iAO indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+

```



```
///| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}
```


iATR

The function returns the handle of the Average True Range indicator. It has only one buffer.

```
int iATR(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    int         ma_period    // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] The value of the averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iATR.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iATR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iATR
#property indicator_label1  "iATR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```



```

#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iATR, // use iATR
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input int          atr_period=14;           // period of calculation
input Creation     type=Call_iATR;          // type of the function
input string       symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double            iATRBuffer[];
//--- variable for storing the handle of the iAC indicator
int              handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average True Range indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iATRBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iATR)
        handle=iATR(name,period,atr_period);
    else
    {
        //--- fill the structure with parameters of the indicator

```



```

    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=atr_period;
    handle=IndicatorCreate(name,period,IND_ATR,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iATR indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Average True Range indicator is calculated for
short_name=StringFormat("iATR(%s/%s, period=%d)",name,EnumToString(period),atr_peri
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iATR indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of val
    //---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {

```



```

    //--- if the iATRBuffer array is greater than the number of values in the iATR
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and s
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the iATRBuffer array with values of the Average True Range indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
    if(!FillArrayFromBuffer(iATRBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Average True Range indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iATR indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer for ATR values
                        int ind_handle,    // handle of the iATR indicator
                        int amount        // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iATRBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iATR indicator, error code %d",GetLast
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |

```



```
//+-----+  
void OnDeinit(const int reason)  
{  
    ///--- clear the chart after deleting the indicator  
    Comment("");  
}
```


iBearsPower

The function returns the handle of the Bears Power indicator. It has only one buffer.

```
int iBearsPower(
    string      symbol,           // symbol name
    ENUM_TIMEFRAMES period,      // period
    int         ma_period,       // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] The value of the averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iBearsPower.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBearsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iBearsPower plot
#property indicator_label1  "iBearsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iBearsPower,      // use iBearsPower
    Call_IndicatorCreate   // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iBearsPower; // type of the function
input int           ma_period=13;          // period of moving average
input string        symbol=" ";            // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double             iBearsPowerBuffer[];
//--- variable for storing the handle of the iBearsPower indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bears Power indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iBearsPowerBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iBearsPower)
        handle=iBearsPower(name,period,ma_period);
    else
    {
        //--- fill the structure with parameters of the indicator

```



```

    MqlParam pars[1];
    //--- period of ma
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_BEARS,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iBearsPower indicator for the symbol
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Bears Power indicator is calculated for
short_name=StringFormat("iBearsPower(%s/%s, period=%d)",name,EnumToString(period),ma_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iBearsPower indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)

```



```

    {
        //--- if the iBearsPowerBuffer array is greater than the number of values in the
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else                        values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

    //--- fill the iBearsPowerBuffer array with values of the Bears Power indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iBearsPowerBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);

    //--- display the service message on the chart
    Comment(comm);

    //--- memorize the number of values in the Bears Power indicator
    bars_calculated=calculated;

    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iBearsPower indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer for Bears Power values
                        int ind_handle,    // handle of the iBearsPower indicator
                        int amount         // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();

    //--- fill a part of the iBearsPowerBuffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iBearsPower indicator, error code %d",
                    GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

    //--- everything is fine
    return(true);
}

//+-----+

```



```
///| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}
```


iBands

The function returns the handle of the Bollinger Bands® indicator.

```
int iBands(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             bands_period,     // period for average line calculation
    int             bands_shift,      // horizontal shift of the indicator
    double          deviation,        // number of standard deviations
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

bands_period

[in] The averaging period of the main line of the indicator.

bands_shift

[in] The shift the indicator relative to the price chart.

deviation

[in] Deviation from the main line.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - BASE_LINE, 1 - UPPER_BAND, 2 - LOWER_BAND

Example:

```
//+-----+
//|                                     Demo_iBands.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
```



```

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBands technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 3
//--- the Upper plot
#property indicator_label1 "Upper"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Lower plot
#property indicator_label2 "Lower"
#property indicator_type2  DRAW_LINE
#property indicator_color2  clrMediumSeaGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- the Middle plot
#property indicator_label3 "Middle"
#property indicator_type3  DRAW_LINE
#property indicator_color3  clrMediumSeaGreen
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iBands,          // use iBands
    Call_IndicatorCreate   // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iBands;          // type of the function
input int               bands_period=20;           // period of moving average
input int               bands_shift=0;             // shift
input double            deviation=2.0;             // number of standard deviations
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string            symbol=" ";               // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffers
double      UpperBuffer[];
double      LowerBuffer[];

```



```

double      MiddleBuffer[];
//--- variable for storing the handle of the iBands indicator
int      handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bollinger Bands indicator
int      bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- assignment of arrays to indicator buffers
SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
SetIndexBuffer(2,MiddleBuffer,INDICATOR_DATA);
//--- set shift of each line
PlotIndexSetInteger(0,PLOT_SHIFT,bands_shift);
PlotIndexSetInteger(1,PLOT_SHIFT,bands_shift);
PlotIndexSetInteger(2,PLOT_SHIFT,bands_shift);
//--- determine the symbol the indicator is drawn for
name=symbol;
//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
//--- take the symbol of the chart the indicator is attached to
name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iBands)
    handle=iBands(name,period,bands_period,bands_shift,deviation,applied_price);
else
{
//--- fill the structure with parameters of the indicator
MqlParam pars[4];
//--- period of ma
pars[0].type=TYPE_INT;
pars[0].integer_value=bands_period;
//--- shift
pars[1].type=TYPE_INT;
pars[1].integer_value=bands_shift;
//--- number of standard deviation
pars[2].type=TYPE_DOUBLE;
pars[2].double_value=deviation;

```



```

    //--- type of price
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_BANDS,4,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iBands indicator for the symbol %s/%s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Bollinger Bands indicator is calculated for
short_name=StringFormat("iBands(%s/%s, %d,%d,%G,%s)",name,EnumToString(period),
    bands_period,bands_shift,deviation,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
    const int prev_calculated,
    const datetime &time[],
    const double &open[],
    const double &high[],
    const double &low[],
    const double &close[],
    const long &tick_volume[],
    const long &volume[],
    const int &spread[])
{
    //--- number of values copied from the iBands indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated

```



```

    {
        //--- if the size of indicator buffers is greater than the number of values in t
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
else
    {
        //--- it means that it's not the first time of the indicator calculation, and s
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- fill the array with values of the Bollinger Bands indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
if(!FillArraysFromBuffers(MiddleBuffer,UpperBuffer,LowerBuffer,bands_shift,handle,v
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                        short_name,
                        values_to_copy);

//--- display the service message on the chart
Comment(comm);

//--- memorize the number of values in the Bollinger Bands indicator
bars_calculated=calculated;

//--- return the prev_calculated value for the next call
return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iBands indicator |
//+-----+

bool FillArraysFromBuffers(double &base_values[], // indicator buffer of the midd
                        double &upper_values[], // indicator buffer of the upper
                        double &lower_values[], // indicator buffer of the lower
                        int shift, // shift
                        int ind_handle, // handle of the iBands indicato
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();

    //--- fill a part of the MiddleBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-shift,amount,base_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
}

```



```

//--- fill a part of the UpperBuffer array with values from the indicator buffer that
if(CopyBuffer(ind_handle,1,-shift,amount,upper_values)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not calculated
    return(false);
}

//--- fill a part of the LowerBuffer array with values from the indicator buffer that
if(CopyBuffer(ind_handle,2,-shift,amount,lower_values)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not calculated
    return(false);
}

//--- everything is fine
return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iBullsPower

The function returns the handle of the Bulls Power indicator. It has only one buffer.

```
int iBullsPower(
    string      symbol,           // symbol name
    ENUM_TIMEFRAMES period,       // period
    int         ma_period,        // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] The averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iBullsPower.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBullsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iBullsPower plot
#property indicator_label1  "iBullsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iBullsPower,      // use iBullsPower
    Call_IndicatorCreate    // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iBullsPower; // type of the function
input int           ma_period=13;          // period of moving average
input string        symbol=" ";            // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double             iBullsPowerBuffer[];
//--- variable for storing the handle of the iBullsPower indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bulls Power indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iBullsPowerBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iBullsPower)
        handle=iBullsPower(name,period,ma_period);
    else
    {
        //--- fill the structure with parameters of the indicator

```



```

    MqlParam pars[1];
    //--- period of ma
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_BULLS,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iBullsPower indicator for the symbol
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Bulls Power indicator is calculated for
short_name=StringFormat("iBullsPower(%s/%s, period=%d)",name,EnumToString(period),period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iBullsPower indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated

```



```

    {
        //--- if the iBullsPowerBuffer array is greater than the number of values in the
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else                        values_to_copy=calculated;
    }
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the iBullsPowerBuffer array with values of the Bulls Power indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iBullsPowerBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);

//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Bulls Power indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iBullsPower indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Bulls Power values
                        int ind_handle,    // handle of the iBullsPower indicator
                        int amount         // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iBullsPowerBuffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iBullsPower indicator, error code %d",
                    //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
}

//--- everything is fine
return(true);
}

//+-----+

```



```
///| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}
//+-----+
```


iCCI

The function returns the handle of the Commodity Channel Index indicator. It has only one buffer.

```
int iCCI(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] The averaging period for the indicator calculation.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iCCI.mq5 |
//|               Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iCCI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"

#property indicator_separate_window
#property indicator_buffers 1
```



```

#property indicator_plots 1
//--- the iCCI plot
#property indicator_label1 "iCCI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- horizontal levels in the indicator window
#property indicator_level1 -100.0
#property indicator_level2 100.0
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iCCI,          // use iCCI
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iCCI;          // type of the function
input int           ma_period=14;            // period of moving average
input ENUM_APPLIED_PRICE applied_price=PRICE_TYPICAL; // type of price
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iCCIBuffer[];
//--- variable for storing the handle of the iCCI indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Commodity Channel Index indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iCCIBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {

```



```

    ///--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
///--- create handle of the indicator
if(type==Call_iCCI)
    handle=iCCI(name,period,ma_period,applied_price);
else
{
    ///--- fill the structure with parameters of the indicator
    MqlParam pars[2];
    ///--- period of moving average
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    ///--- type of price
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_CCI,2,pars);
}
///--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    ///--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iCCI indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    ///--- the indicator is stopped early
    return(INIT_FAILED);
}
///--- show the symbol/timeframe the CCI indicator is calculated for
short_name=StringFormat("iCCI(%s/%s, %d, %s)",name,EnumToString(period),
                        ma_period,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
///--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- number of values copied from the iCCI indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to calculate the indicator for two or more bars (it means that the indicator is not calculated yet)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iCCIBuffer array is greater than the number of values in the iCCI indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so we copy only new values
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the iCCIBuffer array with values of the Commodity Channel Index indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iCCIBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Commodity Channel Index indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iCCI indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Commodity Channel Index
                        int ind_handle,    // handle of the iCCI indicator
                        int amount        // number of copied values
                        )
{
//--- reset error code

```



```

    ResetLastError();
    ///--- fill a part of the iCCIBuffer array with values from the indicator buffer that h
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        ///--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iCCI indicator, error code %d",GetLast
        ///--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    ///--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    ///--- clear the chart after deleting the indicator
    Comment("");
}

```


iChaikin

The function returns the handle of the Chaikin Oscillator indicator. It has only one buffer.

```
int iChaikin(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             fast_ma_period,  // fast period
    int             slow_ma_period,  // slow period
    ENUM_MA_METHOD  ma_method,       // smoothing type
    ENUM_APPLIED_VOLUME applied_volume // type of volume
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

fast_ma_period

[in] Fast averaging period for calculations.

slow_ma_period

[in] Slow averaging period for calculations.

ma_method

[in] Smoothing type. Can be one of the averaging constants of [ENUM_MA_METHOD](#).

applied_volume

[in] The volume used. Can be one of the constants of [ENUM_APPLIED_VOLUME](#).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iChaikin.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```



```

#property description "of indicator buffers for the iChaikin technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iChaikin plot
#property indicator_label1 "iChaikin"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iChaikin,          // use iChaikin
    Call_IndicatorCreate    // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iChaikin;          // type of the function
input int               fast_ma_period=3;            // period of fast ma
input int               slow_ma_period=10;           // period of slow ma
input ENUM_MA_METHOD    ma_method=MODE_EMA;          // type of smoothing
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string            symbol=" ";                  // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;       // timeframe
//--- indicator buffer
double          iChaikinBuffer[];
//--- variable for storing the handle of the iChaikin indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Chaikin Oscillator indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iChaikinBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;

```



```

//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iChaikin)
    handle=iChaikin(name,period,fast_ma_period,slow_ma_period,ma_method,applied_volume);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[4];
    //--- period of fast ma
    pars[0].type=TYPE_INT;
    pars[0].integer_value=fast_ma_period;
    //--- period of slow ma
    pars[1].type=TYPE_INT;
    pars[1].integer_value=slow_ma_period;
    //--- type of smoothing
    pars[2].type=TYPE_INT;
    pars[2].integer_value=ma_method;
    //--- type of volume
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_CHAIKIN,4,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iChaikin indicator for the symbol %s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Chaikin Oscillator indicator is calculated for
short_name=StringFormat("iChaikin(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
    fast_ma_period,slow_ma_period,
    EnumToString(ma_method),EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

```



```

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iChaikin indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iChaikinBuffer array is greater than the number of values in the indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the iChaikinBuffer array with values of the Chaikin Oscillator indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iChaikinBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Chaikin Oscillator indicator

```



```

    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iChaikin indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Chaikin Oscillator
                        int ind_handle,    // handle of the iChaikin indicator
                        int amount        // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iChaikinBuffer array with values from the indicator buffer th
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iChaikin indicator, error code %d",Get
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iCustom

The function returns the handle of a specified custom indicator.

```
int iCustom(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    string      name         // folder/custom_indicator_name
    ...         // list of indicator input parameters
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

name

[in] The name of the custom indicator, with path relative to the root directory of indicators (MQL5/Indicators/). If an indicator is located in a subdirectory, for example, in MQL5/Indicators/Examples, its name must be specified like: "Examples\\indicator_name" (it is necessary to use a double slash instead of the single slash as a separator).

...

[in] [input-parameters](#) of a custom indicator, separated by commas. Type and order of parameters must match. If there is no parameters specified, then [default values](#) will be used.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

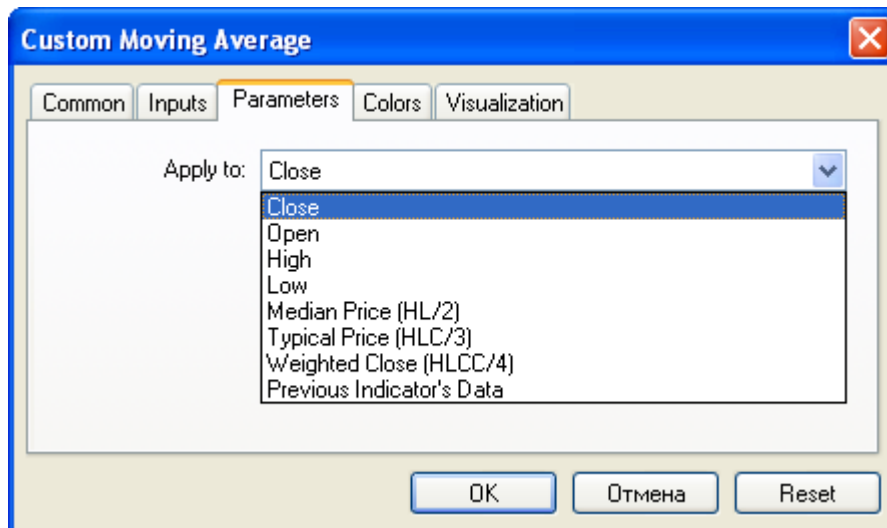
Note

A custom indicator must be compiled (with extension EX5) and located in the directory MQL5/Indicators of the client terminal or its subdirectory.

Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a [constant string](#). For all other cases (use of the [IndicatorCreate\(\)](#) function or use of a non-constant string in the parameter that sets the indicator name) the property [#property tester_indicator](#) is required:

```
#property tester_indicator "indicator_name.ex5"
```

If [the first call form](#) is used in the indicator, then at the custom indicator start you can additionally indicate data for calculation in its "Parameters" tab. If the "Apply to" parameter is not selected explicitly, the default calculation is based on the values of "Close" prices.



When you call a custom indicator from mql5-program, the `Applied_Price` parameter or a handle of another indicator should be passed last, after all input variables of the custom indicator.

See also

[Program Properties](#), [Timeseries and Indicators Access](#), [IndicatorCreate\(\)](#), [IndicatorRelease\(\)](#)

Example:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int MA_Period=21;
input int MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMA;
//--- indicator buffers
double Label1Buffer[];
//--- Handle of the Custom Moving Average.mq5 custom indicator
int MA_handle;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
ResetLastError();
MA_handle=iCustom(NULL,0,"Examples\\Custom Moving Average",
MA_Period,
```



```

        MA_Shift,
        MA_Method,
        PRICE_CLOSE // using the close prices
    );

    Print("MA_handle = ",MA_handle," error = ",GetLastError());
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- Copy the values of the indicator Custom Moving Average to our indicator buffer
    int copy=CopyBuffer(MA_handle,0,0,rates_total,Label1Buffer);
    Print("copy = ",copy," rates_total = ",rates_total);
//--- If our attempt has failed - Report this
    if(copy<=0)
        Print("An attempt to get the values if Custom Moving Average has failed");
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+

```


iDEMA

The function returns the handle of the Double Exponential Moving Average indicator. It has only one buffer.

```
int iDEMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    int             ma_shift,         // horizontal shift
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

ma_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iDEMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```



```

#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iDEMA plot
#property indicator_label1 "iDEMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iDEMA,          // use iDEMA
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iDEMA;          // type of the function
input int           ma_period=14;              // period of moving average
input int           ma_shift=0;                // shift
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";               // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double            iDEMABuffer[];
//--- variable for storing the handle of the iDEMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Double Exponential Moving Average indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iDEMABuffer,INDICATOR_DATA);
    //--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left

```



```

StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iDEMA)
    handle=iDEMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[3];
    //--- period of moving average
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- shift
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- type of price
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_DEMA,3,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iDEMA indicator for the symbol %s/%s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Double Exponential Moving Average indicator is calculated on
short_name=StringFormat("iDEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
    ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
    const int prev_calculated,

```



```

        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
        //--- number of values copied from the iDEMA indicator
        int values_to_copy;
        //--- determine the number of values calculated in the indicator
        int calculated=BarsCalculated(handle);
        if(calculated<=0)
        {
            PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
            return(0);
        }
        //--- if it is the first start of calculation of the indicator or if the number of values
        //---or if it is necessary to calculate the indicator for two or more bars (it means
        if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
        {
            //--- if the iDEMABuffer array is greater than the number of values in the iDEMA
            //--- otherwise, we copy less than the size of indicator buffers
            if(calculated>rates_total) values_to_copy=rates_total;
            else
                values_to_copy=calculated;
        }
        else
        {
            //--- it means that it's not the first time of the indicator calculation, and so
            //--- for calculation not more than one bar is added
            values_to_copy=(rates_total-prev_calculated)+1;
        }
        //--- fill the iDEMABuffer array with values of the Double Exponential Moving Average
        //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
        if(!FillArrayFromBuffer(iDEMABuffer,ma_shift,handle,values_to_copy)) return(0);
        //--- form the message
        string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);
        //--- display the service message on the chart
        Comment(comm);
        //--- memorize the number of values in the Double Exponential Moving Average indicator
        bars_calculated=calculated;
        //--- return the prev_calculated value for the next call
        return(rates_total);
    }
    //+-----+

```



```

//| Filling indicator buffers from the iDEMA indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Double Exponential
                        int shift,         // shift
                        int ind_handle,    // handle of the iDEMA indicator
                        int amount         // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iDEMABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iDEMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iDeMarker

The function returns the handle of the DeMarker indicator. It has only one buffer.

```
int iDeMarker(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    int         ma_period    // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iDeMarker.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDeMarker technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iDeMarker plot
#property indicator_label1  "iDeMarker"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```



```

#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- horizontal levels in the indicator window
#property indicator_level1  0.3
#property indicator_level2  0.7
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iDeMarker,          // use iDeMarker
    Call_IndicatorCreate     // use IndicatorCreate
};
//--- input parameters
input Creation              type=Call_iDeMarker;      // type of the function
input int                   ma_period=14;             // period of moving average
input string                 symbol=" ";              // symbol
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;    // timeframe
//--- indicator buffer
double                      iDeMarkerBuffer[];
//--- variable for storing the handle of the iDeMarker indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the DeMarker indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iDeMarkerBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iDeMarker)
        handle=iDeMarker(name,period,ma_period);
}

```



```

else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    //--- period of moving average
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_DEMARKER,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iDeMarker indicator for the symbol %s",
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the DeMarker indicator is calculated for
short_name=StringFormat("iDeMarker(%s/%s, period=%d)",name,EnumToString(period),ma_
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iDeMarker indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
}

```



```

//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
        //--- if the iDeMarkerBuffer array is greater than the number of values in the i
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the iDeMarkerBuffer array with values of the DeMarker indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
    if(!FillArrayFromBuffer(iDeMarkerBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the DeMarker indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iDeMarker indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer for DeMarker values
                        int ind_handle, // handle of the iDeMarker indicator
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
//--- fill a part of the iDeMarkerBuffer array with values from the indicator buffer t
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iDeMarker indicator, error code %d",Ge
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
//--- everything is fine

```



```
        return(true);
    }
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //-- clear the chart after deleting the indicator
    Comment("");
}
```


iEnvelopes

The function returns the handle of the Envelopes indicator.

```
int iEnvelopes(
    string            symbol,           // symbol name
    ENUM_TIMEFRAMES   period,          // period
    int               ma_period,        // period for the average line calculation
    int               ma_shift,         // horizontal shift of the indicator
    ENUM_MA_METHOD     ma_method,       // type of smoothing
    ENUM_APPLIED_PRICE applied_price,    // type of price or handle
    double            deviation         // deviation of boundaries from the midline
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma period

[in] Averaging period for the main line.

ma shift

[in] The shift of the indicator relative to the price chart.

```
ma method
```

[in] Smoothing type. Can be one of the values of `ENUM_MA_METHOD`.

applied price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

deviation

[in] The deviation from the main line (in percents).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - UPPER_LINE, 1 - LOWER_LINE.

Example:


```

//|                                     Demo_iEnvelopes.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iEnvelopes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 2
//--- the Upper plot
#property indicator_label1 "Upper"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Lower plot
#property indicator_label2 "Lower"
#property indicator_type2  DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iEnvelopes,      // use iEnvelopes
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iEnvelopes;      // type of the function
input int           ma_period=14;              // period of moving average
input int           ma_shift=0;                // shift
input ENUM_MA_METHOD ma_method=MODE_SMA;       // type of smoothing
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input double        deviation=0.1;             // deviation of borders from the
input string        symbol=" ";               // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double      UpperBuffer[];
double      LowerBuffer[];
//--- variable for storing the handle of the iEnvelopes indicator

```



```

int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Envelopes indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
//--- set shift of each line
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
    PlotIndexSetInteger(1,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iEnvelopes)
        handle=iEnvelopes(name,period,ma_period,ma_shift,ma_method,applied_price,deviat:
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[5];
        //--- period of moving average
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        //--- shift
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ma_shift;
        //--- type of smoothing
        pars[2].type=TYPE_INT;
        pars[2].integer_value=ma_method;
        //--- type of price
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        //--- type of price
    }
}

```



```

        pars[4].type=TYPE_DOUBLE;
        pars[4].double_value=deviation;
        handle=IndicatorCreate(name,period,IND_ENVELOPES,5,pars);
    }
//--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iEnvelopes indicator for the symbol
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
//--- show the symbol/timeframe the Envelopes indicator is calculated for
    short_name=StringFormat("iEnvelopes(%s/%s, %d, %d, %s,%s, %G)",name,EnumToString(pe
    ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price),deviation);
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- number of values copied from the iEnvelopes indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {

```



```

    //--- if the UpperBuffer array is greater than the number of values in the iEnvelope
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- fill the UpperBuffer and LowerBuffer arrays with values from the Envelopes indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArraysFromBuffers(UpperBuffer,LowerBuffer,ma_shift,handle,values_to_copy))
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Envelopes indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iEnvelope indicator |
//+-----+
bool FillArraysFromBuffers(double &upper_values[], // indicator buffer of the upper
                        double &lower_values[], // indicator of the lower border
                        int shift, // shift
                        int ind_handle, // handle of the iEnvelope indicator
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the UpperBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-shift,amount,upper_values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iEnvelope indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- fill a part of the LowerBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,1,-shift,amount,lower_values)<0)
    {

```



```
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iEnvelopes indicator, error code %d", GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not copied
    return(false);
}

//--- everything is fine
return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iForce

The function returns the handle of the Force Index indicator. It has only one buffer.

```
int iForce(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    ENUM_MA_METHOD  ma_method,        // smoothing type
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period for the indicator calculations.

ma_method

[in] Smoothing type. Can be one of the values of [ENUM_MA_METHOD](#).

applied_volume

[in] The volume used. Can be one of the values of [ENUM_APPLIED_VOLUME](#).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iForce.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iForce technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
```



```

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- drawing iForce
#property indicator_label1 "iForce"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iForce,          // use iForce
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iForce;          // type of the function
input int               ma_period=13;              // period of averaging
input ENUM_MA_METHOD    ma_method=MODE_SMA;        // type of smoothing
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string            symbol=" ";                // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double          iForceBuffer[];
//--- variable for storing the handle of the iForce indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Force indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iForceBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)

```



```

    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
if(type==Call_iForce)
    handle=iForce(name,period,ma_period,ma_method,applied_volume);
else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[3];
        //--- period of moving average
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        //--- type of smoothing
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ma_method;
        //--- type of volume
        pars[2].type=TYPE_INT;
        pars[2].integer_value=applied_volume;
        //--- type of price
        handle=IndicatorCreate(name,period,IND_FORCE,3,pars);
    }
//--- if the handle is not created
if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iForce indicator for the symbol %s/%s",
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
//--- show the symbol/timeframe the Force indicator is calculated for
short_name=StringFormat("iForce(%s/%s, %d, %s, %s)",name,EnumToString(period),
                        ma_period,EnumToString(ma_method),EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],

```



```

        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])

{
//--- number of values copied from the iForce indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values calculated
//---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iForceBuffer array is greater than the number of values in the iForce indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the iForceBuffer array with values of the Force indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iForceBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Force indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iForce indicator |
//+-----+

bool FillArrayFromBuffer(double &values[], // indicator buffer of Force Index values

```



```

        int ind_handle,    // handle of the iForce indicator
        int amount        // number of copied values
    )

    {
//--- reset error code
    ResetLastError();
//--- fill a part of the iForceBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iForce indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not copied
        return(false);
    }
//--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iFractals

The function returns the handle of the Fractals indicator.

```
int iFractals(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period   // period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - UPPER_LINE, 1 - LOWER_LINE.

Example:

```
//+-----+
//|                                     Demo_iFractals.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFractals technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 2
//--- the FractalUp plot
```



```

#property indicator_label1 "FractalUp"
#property indicator_type1 DRAW_ARROW
#property indicator_color1 clrBlue
//--- the FractalDown plot
#property indicator_label2 "FractalDown"
#property indicator_type2 DRAW_ARROW
#property indicator_color2 clrRed
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iFractals,          // use iFractals
    Call_IndicatorCreate     // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iFractals;          // type of the function
input string        symbol=" ";                  // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;     // timeframe
//--- indicator buffers
double      FractalUpBuffer[];
double      FractalDownBuffer[];
//--- variable for storing the handle of the iFractals indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Fractals indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,FractalUpBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,FractalDownBuffer,INDICATOR_DATA);
    //--- set codes using a symbol from the Wingdings charset for the PLOT_ARROW property
    PlotIndexSetInteger(0,PLOT_ARROW,217); // arrow up
    PlotIndexSetInteger(1,PLOT_ARROW,218); // arrow down
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {

```



```

    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iFractals)
    handle=iFractals(name,period);
else
    handle=IndicatorCreate(name,period,IND_FRACTALS);
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iFractals indicator for the symbol %s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Fractals indicator is calculated for
short_name=StringFormat("iFractals(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iFractals indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of val

```



```

//---or if it is necessary to calculated the indicator for two or more bars (it means
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- if the FractalUpBuffer array is greater than the number of values in the iFractals indicator
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the FractalUpBuffer and FractalDownBuffer arrays with values from the iFractals indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
if(!FillArraysFromBuffers(FractalUpBuffer,FractalDownBuffer,handle,values_to_copy))
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                        short_name,
                        values_to_copy);

//--- display the service message on the chart
Comment(comm);

//--- memorize the number of values in the Fractals indicator
bars_calculated=calculated;

//--- return the prev_calculated value for the next call
return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iFractals indicator |
//+-----+

bool FillArraysFromBuffers(double &up_arrows[], // indicator buffer for up arrows
                        double &down_arrows[], // indicator buffer for down arrows
                        int ind_handle, // handle of the iFractals indicator
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();

    //--- fill a part of the FractalUpBuffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,0,0,amount,up_arrows)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iFractals indicator to the FractalUpBuffer array\n");
        GetLastError();

        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
}

```



```

//--- fill a part of the FractalDownBuffer array with values from the indicator buffer
if(CopyBuffer(ind_handle,1,0,amount,down_arrows)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iFractals indicator to the FractalDownBuffer array\n");
    GetLastError();
    //--- quit with zero result - it means that the indicator is considered as not connected
    return(false);
}
//--- everything is fine
return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iFrAMA

The function returns the handle of the Fractal Adaptive Moving Average indicator. It has only one buffer.

```
int iFrAMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    int             ma_shift,         // horizontal shift on the chart
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Period (bars count) for the indicator calculations.

ma_shift

[in] Shift of the indicator in the price chart.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iFrAMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFrAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```



```

#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- drawing iFrAMA
#property indicator_label1 "iFrAMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iFrAMA,          // use iFrAMA
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iFrAMA;          // type of the function
input int               ma_period=14;              // period of averaging
input int               ma_shift=0;                // shift
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string            symbol=" ";                // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffer
double iFrAMABuffer[];
//--- variable for storing the handle of the iFrAMA indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Fractal Adaptive Moving Average indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iFrAMABuffer,INDICATOR_DATA);
    //--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left

```



```

StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iFrAMA)
    handle=iFrAMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[3];
    //--- period of moving average
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- shift
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- type of price
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    //--- type of price
    handle=IndicatorCreate(name,period,IND_FRAMA,3,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iFrAMA indicator for the symbol %s/%s",
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the iFrAMA indicator is calculated for
short_name=StringFormat("iFrAMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                        ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,

```



```

        const int prev_calculated,
        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])

{
    //--- number of values copied from the iFrAMA indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iFrAMABuffer array is greater than the number of values in the iFrAMA
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the iFrAMABuffer array with values of the Fractal Adaptive Moving Average
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iFrAMABuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                               TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                               short_name,
                               values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Fractal Adaptive Moving Average indicator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

```



```

//+-----+
//| Filling indicator buffers from the iFrAMA indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Fractal Adaptive Mo
                        int shift,        // shift
                        int ind_handle,    // handle of the iFrAMA indicator
                        int amount        // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iFrAMABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iFrAMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iGator

The function returns the handle of the Gator indicator. The Oscillator shows the difference between the blue and red lines of Alligator (upper histogram) and difference between red and green lines (lower histogram).

```
int iGator(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             jaw_period,      // period for the calculation of the jaws
    int             jaw_shift,       // jaws horizontal shift
    int             teeth_period,    // period for the calculation of the teeth
    int             teeth_shift,     // teeth horizontal shift
    int             lips_period,     // period for the calculation of the lips
    int             lips_shift,      // lips horizontal shift
    ENUM_MA_METHOD  ma_method,      // type of smoothing
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

jaw_period

[in] Averaging period for the blue line (Alligator's Jaw).

jaw_shift

[in] The shift of the blue line relative to the price chart. It isn't directly connected with the visual shift of the indicator histogram.

teeth_period

[in] Averaging period for the red line (Alligator's Teeth).

teeth_shift

[in] The shift of the red line relative to the price chart. It isn't directly connected with the visual shift of the indicator histogram.

lips_period

[in] Averaging period for the green line (Alligator's lips).

lips_shift

[in] The shift of the green line relative to the price charts. It isn't directly connected with the visual shift of the indicator histogram.

ma_method

[in] Smoothing type. Can be one of the values of [ENUM_MA_METHOD](#).

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

Buffer numbers: 0 - UPPER_HISTOGRAM, 1 - color buffer of the upper histogram, 2 - LOWER_HISTOGRAM, 3 - color buffer of the lower histogram.

Example:

```
//+-----+
//|                                     Demo_iGator.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iGator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All other parameters are as in a standard Gator Oscillator."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 2
//--- drawing GatorUp
#property indicator_label1 "GatorUp"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- drawing GatorDown
#property indicator_label2 "GatorDown"
#property indicator_type2  DRAW_COLOR_HISTOGRAM
#property indicator_color2  clrGreen, clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+

enum Creation
```



```

{
    Call_iGator,          // use iGator
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation           type=Call_iGator;      // type of the function
input string             symbol=" ";            // symbol
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // timeframe
input int                jaw_period=13;         // period of the Jaw line
input int                jaw_shift=8;           // shift of the Jaw line
input int                teeth_period=8;        // period of the Teeth line
input int                teeth_shift=5;         // shift of the Teeth line
input int                lips_period=5;         // period of the Lips line
input int                lips_shift=3;          // shift of the Lips line
input ENUM_MA_METHOD      MA_method=MODE_SMMMA; // method of averaging of the Alligator
input ENUM_APPLIED_PRICE  applied_price=PRICE_MEDIAN; // type of price used for calculation
//--- indicator buffers
double      GatorUpBuffer[];
double      GatorUpColors[];
double      GatorDownBuffer[];
double      GatorDownColors[];
//--- variable for storing the handle of the iGator indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- shift values for the upper and lower histograms
int shift;
//--- we will keep the number of values in the Gator Oscillator indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,GatorUpBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,GatorUpColors,INDICATOR_COLOR_INDEX);
    SetIndexBuffer(2,GatorDownBuffer,INDICATOR_DATA);
    SetIndexBuffer(3,GatorDownColors,INDICATOR_COLOR_INDEX);
/*
    All the shifts specified in the parameters refer to the Alligator indicator on the chart.
    That's is why they don't move the Gator indicator itself, but they move the Alligator
    which values are used for calculation of the Gator Oscillator!
*/
//--- let's calculate the shift for the upper and lower histograms, that is equal to the
    shift=MathMin(jaw_shift,teeth_shift);
    PlotIndexSetInteger(0,PLOT_SHIFT,shift);

```



```

//--- despite the indicator contains two histograms, the same shift is used - this is
    PlotIndexSetInteger(1,PLOT_SHIFT,shift);

//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iGator)
        handle=iGator(name,period,jaw_period,jaw_shift,teeth_period,teeth_shift,
            lips_period,lips_shift,MA_method,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[8];
        //--- periods and shifts of the Alligator lines
        pars[0].type=TYPE_INT;
        pars[0].integer_value=jaw_period;
        pars[1].type=TYPE_INT;
        pars[1].integer_value=jaw_shift;
        pars[2].type=TYPE_INT;
        pars[2].integer_value=teeth_period;
        pars[3].type=TYPE_INT;
        pars[3].integer_value=teeth_shift;
        pars[4].type=TYPE_INT;
        pars[4].integer_value=lips_period;
        pars[5].type=TYPE_INT;
        pars[5].integer_value=lips_shift;
        //--- type of smoothing
        pars[6].type=TYPE_INT;
        pars[6].integer_value=MA_method;
        //--- type of price
        pars[7].type=TYPE_INT;
        pars[7].integer_value=applied_price;
        //--- create handle
        handle=IndicatorCreate(name,period,IND_GATOR,8,pars);
    }
//--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iGator indicator for the symbol %s/");
    }

```



```

        name,
        EnumToString(period),
        GetLastError());

    //--- the indicator is stopped early
    return(INIT_FAILED);
}

//--- show the symbol/timeframe the Gator Oscillator indicator is calculated for
short_name=StringFormat("iGator(%s/%s, %d, %d, %d, %d, %d, %d)",name,EnumToString(p
        jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,l
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iGator indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of val
    //---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
        //--- if the GatorUpBuffer array is greater than the number of values in the iG
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s
        //--- for calculation not more than one bar is added

```



```

        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the arrays with values of the Gator Oscillator indicator
//--- if FillArraysFromBuffer returns false, it means the information is not ready yet
    if(!FillArraysFromBuffers(GatorUpBuffer,GatorUpColors,GatorDownBuffer,GatorDownColors,
        shift,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
        short_name,
        values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Gator Oscillator indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iGator indicator |
//+-----+
bool FillArraysFromBuffers(double &ups_buffer[],           // indicator buffer for the upper and lower
                        double &up_color_buffer[],         // indicator buffer for price color
                        double &downs_buffer[],           // indicator buffer for the upper and lower
                        double &downs_color_buffer[],       // indicator buffer for price color
                        int u_shift,                       // shift for the upper and lower
                        int ind_handle,                   // handle of the iGator indicator
                        int amount                        // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the GatorUpBuffer array with values from the indicator buffer the
    if(CopyBuffer(ind_handle,0,-u_shift,amount,ups_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

//--- fill a part of the GatorUpColors array with values from the indicator buffer the
    if(CopyBuffer(ind_handle,1,-u_shift,amount,up_color_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
}

```



```

//--- fill a part of the GatorDownBuffer array with values from the indicator buffer t
if(CopyBuffer(ind_handle,2,-u_shift,amount,downs_buffer)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not c
    return(false);
}

//--- fill a part of the GatorDownColors array with values from the indicator buffer t
if(CopyBuffer(ind_handle,3,-u_shift,amount,downs_color_buffer)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not c
    return(false);
}

//--- everything is fine
return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


ilchimoku

The function returns the handle of the Ichimoku Kinko Hyo indicator.

```
int iIchimoku(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,         // period
    int             tenkan_sen,      // period of Tenkan-sen
    int             kijun_sen,       // period of Kijun-sen
    int             senkou_span_b    // period of Senkou Span B
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

tenkan_sen

[in] Averaging period for Tenkan Sen.

kijun_sen

[in] Averaging period for Kijun Sen.

senkou_span_b

[in] Averaging period for Senkou Span B.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - TENKANSEN_LINE, 1 - KIJUNSEN_LINE, 2 - SENKOUSPANA_LINE, 3 - SENKOUSPANB_LINE, 4 - CHIKOSPAN_LINE.

Example:

```
//+-----+
//|                                     Demo_iIchimoku.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```



```

#property description "of indicator buffers for the iIchimoku technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters just like in the standard Ichimoku Kinko I

#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 4
//--- the Tenkan_sen plot
#property indicator_label1 "Tenkan_sen"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- the Kijun_sen plot
#property indicator_label2 "Kijun_sen"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrBlue
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//--- the Senkou_Span plot
#property indicator_label3 "Senkou Span A;Senkou Span B" // two fields will be shown
#property indicator_type3 DRAW_FILLING
#property indicator_color3 clrSandyBrown, clrThistle
#property indicator_style3 STYLE_SOLID
#property indicator_width3 1
//--- the Chikou_Span plot
#property indicator_label4 "Chinkou_Span"
#property indicator_type4 DRAW_LINE
#property indicator_color4 clrLime
#property indicator_style4 STYLE_SOLID
#property indicator_width4 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iIchimoku,          // use iIchimoku
    Call_IndicatorCreate     // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iIchimoku;          // type of the function
input int               tenkan_sen=9;                 // period of Tenkan-sen
input int               kijun_sen=26;                 // period of Kijun-sen
input int               senkou_span_b=52;            // period of Senkou Span B
input string            symbol=" ";                  // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;       // timeframe
//--- indicator buffer

```



```

double      Tenkan_sen_Buffer[];
double      Kijun_sen_Buffer[];
double      Senkou_Span_A_Buffer[];
double      Senkou_Span_B_Buffer[];
double      Chinkou_Span_Buffer[];
//--- variable for storing the handle of the iIchimoku indicator
int         handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Ichimoku Kinko Hyo indicator
int         bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- assignment of arrays to indicator buffers
SetIndexBuffer(0,Tenkan_sen_Buffer,INDICATOR_DATA);
SetIndexBuffer(1,Kijun_sen_Buffer,INDICATOR_DATA);
SetIndexBuffer(2,Senkou_Span_A_Buffer,INDICATOR_DATA);
SetIndexBuffer(3,Senkou_Span_B_Buffer,INDICATOR_DATA);
SetIndexBuffer(4,Chinkou_Span_Buffer,INDICATOR_DATA);
//--- set the shift for the Senkou Span channel of kijun_sen bars in the future direction
PlotIndexSetInteger(2,PLOT_SHIFT,kijun_sen);
//--- setting a shift for the Chikou Span line is not required, since the Chinkou data
//--- is already stored with a shift in iIchimoku
//--- determine the symbol the indicator is drawn for
name=symbol;
//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
//--- take the symbol of the chart the indicator is attached to
name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iIchimoku)
handle=iIchimoku(name,period,tenkan_sen,kijun_sen,senkou_span_b);
else
{
//--- fill the structure with parameters of the indicator
MqlParam pars[3];
//--- periods and shifts of the Alligator lines
pars[0].type=TYPE_INT;
pars[0].integer_value=tenkan_sen;

```



```

    pars[1].type=TYPE_INT;
    pars[1].integer_value=kijun_sen;
    pars[2].type=TYPE_INT;
    pars[2].integer_value=senkou_span_b;
    //--- create handle
    handle=IndicatorCreate(name,period,IND_ICHIMOKU,3,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iIchimoku indicator for the symbol %s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Ichimoku Kinko Hyo indicator is calculated for
short_name=StringFormat("iIchimoku(%s/%s, %d, %d, %d)",name,EnumToString(period),
    tenkan_sen,kijun_sen,senkou_span_b);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
    const int prev_calculated,
    const datetime &time[],
    const double &open[],
    const double &high[],
    const double &low[],
    const double &close[],
    const long &tick_volume[],
    const long &volume[],
    const int &spread[])
{
    //--- number of values copied from the iIchimoku indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of val

```



```

//---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the Tenkan_sen_Buffer array is greater than the number of values in the
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the arrays with values of the Ichimoku Kinko Hyo indicator
//--- if FillArraysFromBuffer returns false, it means the information is not ready yet
    if(!FillArraysFromBuffers(Tenkan_sen_Buffer,Kijun_sen_Buffer,Senkou_Span_A_Buffer,Senkou_Span_B_Buffer,
        kijun_sen,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
        short_name,
        values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Ichimoku Kinko Hyo indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iIchimoku indicator |
//+-----+

bool FillArraysFromBuffers(double &tenkan_sen_buffer[], // indicator buffer of the
    double &kijun_sen_buffer[], // indicator buffer of the
    double &senkou_span_A_buffer[], // indicator buffer of the
    double &senkou_span_B_buffer[], // indicator buffer of the
    double &chinkou_span_buffer[], // indicator buffer of the
    int senkou_span_shift, // shift of the Senkou Span
    int ind_handle, // handle of the iIchimoku
    int amount // number of copied values
)
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the Tenkan_sen_Buffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,0,0,amount,tenkan_sen_buffer)<0)
    {
        //--- if the copying fails, tell the error code

```



```

        PrintFormat("1.Failed to copy data from the iIchimoku indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- fill a part of the Kijun_sen_Buffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,1,0,amount,kijun_sen_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("2.Failed to copy data from the iIchimoku indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- fill a part of the Chinkou_Span_Buffer array with values from the indicator buff
    //--- if senkou_span_shift>0, the line is shifted in the future direction by senkou_sp
    if(CopyBuffer(ind_handle,2,-senkou_span_shift,amount,senkou_span_A_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("3.Failed to copy data from the iIchimoku indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- fill a part of the Senkou_Span_A_Buffer array with values from the indicator bu
    //--- if senkou_span_shift>0, the line is shifted in the future direction by senkou_sp
    if(CopyBuffer(ind_handle,3,-senkou_span_shift,amount,senkou_span_B_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("4.Failed to copy data from the iIchimoku indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- fill a part of the Senkou_Span_B_Buffer array with values from the indicator bu
    //--- when copying Chinkou Span, we don't need to consider the shift, since the Chinko
    //--- is already stored with a shift in iIchimoku
    if(CopyBuffer(ind_handle,4,0,amount,chinkou_span_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("5.Failed to copy data from the iIchimoku indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |

```



```
//+-----+  
void OnDeinit(const int reason)  
{  
    ///--- clear the chart after deleting the indicator  
    Comment("");  
}
```


iBWMFI

The function returns the handle of the Market Facilitation Index indicator. It has only one buffer.

```
int iBWMFI(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be one of the constants of [ENUM_APPLIED_VOLUME](#).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iBWMFI.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBWMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- the iBWMFI plot
#property indicator_label1 "iBWMFI"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
#property indicator_color1 clrLime,clrSaddleBrown,clrBlue,clrPink
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iBWMFI,          // use iBWMFI
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iBWMFI;          // type of function
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string            symbol=" ";                // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double      iBWMFIBuffer[];
double      iBWMFIColors[];
//--- variable for storing the handle of the iBWMFI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Market Facilitation Index by Bill Williams
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,iBWMFIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iBWMFIColors,INDICATOR_COLOR_INDEX);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iBWMFI)
        handle=iBWMFI(name,period,applied_volume);
    else

```



```

{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    //--- type of volume
    pars[0].type=TYPE_INT;
    pars[0].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_BWMFI,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iBWMFI indicator for the symbol %s/%s",
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Market Facilitation Index by Bill Williams indicator
short_name=StringFormat("iBWMFI(%s/%s, %s)",name,EnumToString(period),
                        EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iBWMFI indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
}

```



```

//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
        //--- if the iBWMFIBuffer array is greater than the number of values in the iBWMFI
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- fill the arrays with values of the indicator Market Facilitation Index by Bill W
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
    if(!FillArraysFromBuffers(iBWMFIBuffer,iBWMFIColors,handle,values_to_copy)) return
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);

//--- display the service message on the chart
    Comment(comm);

//--- memorize the number of values in the Market Facilitation Index by Bill Williams
    bars_calculated=calculated;

//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iBWMFI indicator |
//+-----+

bool FillArraysFromBuffers(double &values[], // indicator buffer for the histogram
                           double &colors[], // indicator buffer of the histogram c
                           int ind_handle, // handle of the iBWMFI indicator
                           int amount // number of copied values
                           )
{
    //--- reset error code
    ResetLastError();

//--- fill a part of the iBWMFIBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLa
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
}

```



```

//--- fill a part of the iBWMFIColors array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,1,0,amount,colors)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
//--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iMomentum

The function returns the handle of the Momentum indicator. It has only one buffer.

```
int iMomentum(
    string      symbol,           // symbol name
    ENUM_TIMEFRAMES period,      // period
    int         mom_period,       // averaging period
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

mom_period

[in] Averaging period (bars count) for the calculation of the price change.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iMomentum.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMomentum technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Momentum."

#property indicator_separate_window
```



```

#property indicator_buffers 1
#property indicator_plots 1
//--- plot iMomentum
#property indicator_label1 "iMomentum"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDodgerBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iMomentum,      // use iMomentum
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iMomentum;      // type of the function
input int               mom_period=14;             // period of Momentum
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string            symbol=" ";               // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffer
double      iMomentumBuffer[];
//--- variable for storing the handle of the iMomentum indicator
int         handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Momentum indicator
int     bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iMomentumBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
}

```



```

    }
    ///--- create handle of the indicator
    if(type==Call_iMomentum)
        handle=iMomentum(name,period,mom_period,applied_price);
    else
    {
        ///--- fill the structure with parameters of the indicator
        MqlParam pars[2];
        ///--- period
        pars[0].type=TYPE_INT;
        pars[0].integer_value=mom_period;
        ///--- type of price
        pars[1].type=TYPE_INT;
        pars[1].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_MOMENTUM,2,pars);
    }
    ///--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        ///--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iMomentum indicator for the symbol %s",
                    name,
                    EnumToString(period),
                    GetLastError());
        ///--- the indicator is stopped early
        return(INIT_FAILED);
    }
    ///--- show the symbol/timeframe the Momentum indicator is calculated for
    short_name=StringFormat("iMomentum(%s/%s, %d, %s)",name,EnumToString(period),
                            mom_period, EnumToString(applied_price));
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
    ///--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ///--- number of values copied from the iMomentum indicator

```



```

    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iMomentumBuffer array is greater than the number of values in the indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the iMomentumBuffer array with values of the Momentum indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iMomentumBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Momentum indicator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iMomentum indicator |
//+-----+

bool FillArrayFromBuffer(double &values[], // indicator buffer of Momentum values
                        int ind_handle,    // handle of the iMomentum indicator
                        int amount         // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iMomentumBuffer array with values from the indicator buffer

```



```
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iMomentum indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not calculated
    return(false);
}
//--- everything is fine
return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iMFI

The function returns the handle of the Money Flow Index indicator.

```
int iMFI(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for the calculation.

applied_volume

[in] The volume used. Can be any of the [ENUM_APPLIED_VOLUME](#) values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iMFI.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Money Flow"

#property indicator_separate_window
#property indicator_buffers 1
```



```

#property indicator_plots 1
//--- the iMFI plot
#property indicator_label1 "iMFI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDodgerBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- horizontal levels in the indicator window
#property indicator_level1 20
#property indicator_level2 80
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iMFI,          // use iMFI
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iMFI;          // type of the function
input int           ma_period=14;            // period
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iMFIBuffer[];
//--- variable for storing the handle of the iMFI indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Money Flow Index indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iMFIBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {

```



```

    ///--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
///--- create handle of the indicator
if(type==Call_iMFI)
    handle=iMFI(name,period,ma_period,applied_volume);
else
{
    ///--- fill the structure with parameters of the indicator
    MqlParam pars[2];
    ///--- period
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    ///--- type of volume
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_MFI,2,pars);
}
///--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    ///--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iMFI indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    ///--- the indicator is stopped early
    return(INIT_FAILED);
}
///--- show the symbol/timeframe the Money Flow Index indicator is calculated for
short_name=StringFormat("iMFI(%s/%s, %d, %s)",name,EnumToString(period),
                        ma_period, EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
///--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- number of values copied from the iMFI indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to calculate the indicator for two or more bars (it means that the indicator is not calculated yet)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iMFIBuffer array is greater than the number of values in the iMFI indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so we need to calculate only for one bar
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the iMFIBuffer array with values of the Money Flow Index indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iMFIBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                              TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                              short_name,
                              values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Money Flow Index indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iMFI indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Money Flow Index values
                        int ind_handle,    // handle of the iMFI indicator
                        int amount         // number of copied values
                        )
{
//--- reset error code

```



```

    ResetLastError();
    ///--- fill a part of the iMFIBuffer array with values from the indicator buffer that h
    if(CopyBuffer(ind_handle,0,0,amount,values)<0)
    {
        ///--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iMFI indicator, error code %d",GetLast
        ///--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    ///--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    ///--- clear the chart after deleting the indicator
    Comment("");
}

```


iMA

The function returns the handle of the Moving Average indicator. It has only one buffer.

```
int iMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    int             ma_shift,         // horizontal shift
    ENUM_MA_METHOD  ma_method,        // smoothing type
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period for the calculation of the moving average.

ma_shift

[in] Shift of the indicator relative to the price chart.

ma_method

[in] Smoothing type. Can be one of the [ENUM_MA_METHOD](#) values.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```



```

#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Moving Average."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iMA plot
#property indicator_label1 "iMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iMA,          // use iMA
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iMA;          // type of the function
input int           ma_period=10;           // period of ma
input int           ma_shift=0;             // shift
input ENUM_MA_METHOD ma_method=MODE_SMA;    // type of smoothing
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";            // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iMABuffer[];
//--- variable for storing the handle of the iMA indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Average indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iMABuffer,INDICATOR_DATA);

```



```

//--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iMA)
        handle=iMA(name,period,ma_period,ma_shift,ma_method,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[4];
        //--- period
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        //--- shift
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ma_shift;
        //--- type of smoothing
        pars[2].type=TYPE_INT;
        pars[2].integer_value=ma_method;
        //--- type of price
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_MA,4,pars);
    }
//--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iMA indicator for the symbol %s/%s,
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
//--- show the symbol/timeframe the Moving Average indicator is calculated for
    short_name=StringFormat("iMA(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                            ma_period, ma_shift,EnumToString(ma_method),EnumToString(applied_price));
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```



```

//--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iMA indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iMABuffer array is greater than the number of values in the iMA indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the iMABuffer array with values of the Moving Average indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iMABuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
}

```



```

//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Moving Average indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the MA indicator |
//+-----+
bool FillArrayFromBuffer(double &values[], // indicator buffer of Moving Average values
                        int shift,        // shift
                        int ind_handle,    // handle of the iMA indicator
                        int amount         // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iMABuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iOsMA

The function returns the handle of the Moving Average of Oscillator indicator. The OsMA oscillator shows the difference between values of MACD and its signal line. It has only one buffer.

```
int iOsMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             fast_ema_period, // period for Fast Moving Average
    int             slow_ema_period, // period for Slow Moving Average
    int             signal_period,   // averaging period for their difference
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

fast_ema_period

[in] Period for Fast Moving Average calculation.

slow_ema_period

[in] Period for Slow Moving Average calculation.

signal_period

[in] Averaging period for signal line calculation.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

In some systems this oscillator is also known as MACD histogram.

Example:

```
//+-----+
//|                                     Demo_iOsMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
```



```

//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOsMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Moving Average"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iOsMA plot
#property indicator_label1  "iOsMA"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iOsMA,          // use iOsMA
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iOsMA;          // type of the function
input int               fast_ema_period=12;        // period of fast ma
input int               slow_ema_period=26;        // period of slow ma
input int               signal_period=9;           // period of averaging of differ
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string            symbol=" ";               // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffer
double          iOsMABuffer[];
//--- variable for storing the handle of the iOsMA indicator
int             handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Average indicator
int             bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+

```



```

int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iOsMABuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iOsMA)
        handle=iOsMA(name,period,fast_ema_period,slow_ema_period,signal_period,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[4];
        //--- period of fast ma
        pars[0].type=TYPE_INT;
        pars[0].integer_value=fast_ema_period;
        //--- period of slow ma
        pars[1].type=TYPE_INT;
        pars[1].integer_value=slow_ema_period;
        //--- period of averaging of difference between the fast and the slow moving average
        pars[2].type=TYPE_INT;
        pars[2].integer_value=signal_period;
        //--- type of price
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_OSMA,4,pars);
    }
    //--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create a handle of iOsMA for the pair %s/%s, error code %d",
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
    //--- show the symbol/timeframe the Moving Average of Oscillator indicator is calculated for
    short_name=StringFormat("iOsMA(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),

```



```

        fast_ema_period,slow_ema_period,signal_period,EnumToString
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- number of values copied from the iOsMA indicator
int values_to_copy;
//--- determine the number of values calculated in the indicator
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
    return(0);
}
//--- if it is the first start of calculation of the indicator or if the number of values
//---or if it is necessary to calculate the indicator for two or more bars (it means
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- if the iOsMABuffer array is greater than the number of values in the iOsMA
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and so
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- fill the arrays with values of the iOsMA indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
if(!FillArrayFromBuffer(iOsMABuffer,handle,values_to_copy)) return(0);
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),

```



```

        short_name,
        values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Moving Average of Oscillator indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iOsMA indicator |
//+-----+
bool FillArrayFromBuffer(double &ama_buffer[], // indicator buffer of OsMA values
                        int ind_handle,        // handle of the iOsMA indicator
                        int amount             // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iOsMABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,ama_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iOsMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iMACD

The function returns the handle of the Moving Averages Convergence/Divergence indicator. In systems where OsMA is called MACD Histogram, this indicator is shown as two lines. In the client terminal the Moving Averages Convergence/Divergence looks like a histogram.

```
int iMACD(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             fast_ema_period,  // period for Fast average calculation
    int             slow_ema_period,  // period for Slow average calculation
    int             signal_period,    // period for their difference averaging
    ENUM_APPLIED_PRICE applied_price  // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

fast_ema_period

[in] Period for Fast Moving Average calculation.

slow_ema_period

[in] Period for Slow Moving Average calculation.

signal_period

[in] Period for Signal line calculation.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+-----+
//|                                     Demo_iMACD.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
```



```

//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMACD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All other parameters like in the standard MACD."

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- the MACD plot
#property indicator_label1 "MACD"
#property indicator_type1  DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Signal plot
#property indicator_label2 "Signal"
#property indicator_type2  DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iMACD,          // use iMACD
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iMACD;          // type of the function
input int           fast_ema_period=12;        // period of fast ma
input int           slow_ema_period=26;        // period of slow ma
input int           signal_period=9;          // period of averaging of differ
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";               // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double      MACDBuffer[];
double      SignalBuffer[];
//--- variable for storing the handle of the iMACD indicator
int         handle;
//--- variable for storing

```



```

string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Averages Convergence/Divergence
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- assignment of arrays to indicator buffers
SetIndexBuffer(0,MACDBuffer,INDICATOR_DATA);
SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
name=symbol;
//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
//--- take the symbol of the chart the indicator is attached to
name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iMACD)
handle=iMACD(name,period,fast_ema_period,slow_ema_period,signal_period,applied_p
else
{
//--- fill the structure with parameters of the indicator
MqlParam pars[4];
//--- period of fast ma
pars[0].type=TYPE_INT;
pars[0].integer_value=fast_ema_period;
//--- period of slow ma
pars[1].type=TYPE_INT;
pars[1].integer_value=slow_ema_period;
//--- period of averaging of difference between the fast and the slow moving ave
pars[2].type=TYPE_INT;
pars[2].integer_value=signal_period;
//--- type of price
pars[3].type=TYPE_INT;
pars[3].integer_value=applied_price;
handle=IndicatorCreate(name,period,IND_MACD,4,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
//--- tell about the failure and output the error code

```



```

        PrintFormat("Failed to create handle of the iMACD indicator for the symbol %s/%s",
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
    //--- show the symbol/timeframe the Moving Average Convergence/Divergence indicator is
    short_name=StringFormat("iMACD(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                            fast_ema_period,slow_ema_period,signal_period,EnumToString
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
    //--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iMACD indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the MACDBuffer array is greater than the number of values in the iMACD
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and s

```



```

        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the arrays with values of the iMACD indicator
    //--- if FillArraysFromBuffer returns false, it means the information is not ready yet
    if(!FillArraysFromBuffers(MACDBuffer,SignalBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Moving Averages indicator Convergence/Divergence
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iMACD indicator |
//+-----+
bool FillArraysFromBuffers(double &macd_buffer[], // indicator buffer of MACD value
                           double &signal_buffer[], // indicator buffer of the signal
                           int ind_handle, // handle of the iMACD indicator
                           int amount // number of copied values
                           )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iMACDBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,0,amount,macd_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

    //--- fill a part of the SignalBuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }

    //--- everything is fine
    return(true);
}

```



```
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iOBV

The function returns the handle of the On Balance Volume indicator. It has only one buffer.

```
int iOBV(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be any of the [ENUM_APPLIED_VOLUME](#) values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iOBV.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOBV technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iOBV
#property indicator_label1  "iOBV"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iOBV ,           // use iOBV
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iOBV;           // type of the function
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string            symbol=" ";               // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffers
double iOBVBuffer[];
//--- variable for storing the handle of the iOBV indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the On Balance Volume indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iOBVBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iOBV)
        handle=iOBV(name,period,applied_volume);
    else
    {
        //--- fill the structure with parameters of the indicator

```



```

    MqlParam pars[1];
    //--- type of volume
    pars[0].type=TYPE_INT;
    pars[0].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_OBV,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iOBV indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the On Balance Volume indicator is calculated for
short_name=StringFormat("iOBV(%s/%s, %s)",name,EnumToString(period),
                        EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iOBV indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of va
    //---or if it is necessary to calculated the indicator for two or more bars (it means

```



```

    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iOBVBuffer array is greater than the number of values in the iOBV indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

    //--- fill the arrays with values of the iOBV indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iOBVBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);

    //--- display the service message on the chart
    Comment(comm);

    //--- memorize the number of values in the On Balance Volume indicator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iOBV indicator |
//+-----+
bool FillArrayFromBuffer(double &obv_buffer[], // indicator buffer of OBV values
                        int ind_handle,        // handle of the iOBV indicator
                        int amount             // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iOBVBuffer array with values from the indicator buffer that has
    if(CopyBuffer(ind_handle,0,0,amount,obv_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iOBV indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}

```



```
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iSAR

The function returns the handle of the Parabolic Stop and Reverse system indicator. It has only one buffer.

```
int iSAR(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    double      step,        // price increment step - acceleration factor
    double      maximum      // maximum value of step
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

step

[in] The step of price increment, usually 0.02.

maximum

[in] The maximum step, usually 0.2.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iSAR.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iSAR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Parabolic"

#property indicator_chart_window
```



```

#property indicator_buffers 1
#property indicator_plots 1
//--- drawing iSAR
#property indicator_label1 "iSAR"
#property indicator_type1 DRAW_ARROW
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iSAR,          // use iSAR
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iSAR;          // type of the function
input double        step=0.02;               // step - the acceleration factor
input double        maximum=0.2;             // maximum value of step
input string         symbol=" ";             // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double iSARBuffer[];
//--- variable for storing the handle of the iSAR indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Parabolic SAR indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iSARBuffer,INDICATOR_DATA);
    //--- set a symbol code from the Wingdings charset for the PLOT_ARROW property for display
    PlotIndexSetInteger(0,PLOT_ARROW,159);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {

```



```

    ///--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
///--- create handle of the indicator
if(type==Call_iSAR)
    handle=iSAR(name,period,step,maximum);
else
{
    ///--- fill the structure with parameters of the indicator
    MqlParam pars[2];
    ///--- step value
    pars[0].type=TYPE_DOUBLE;
    pars[0].double_value=step;
    ///--- limit of the step value that can be used for calculations
    pars[1].type=TYPE_DOUBLE;
    pars[1].double_value=maximum;
    handle=IndicatorCreate(name,period,IND_SAR,2,pars);
}
///--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    ///--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iSAR indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    ///--- the indicator is stopped early
    return(INIT_FAILED);
}
///--- show the symbol/timeframe the Parabolic SAR indicator is calculated for
short_name=StringFormat("iSAR(%s/%s, %G, %G)",name,EnumToString(period),
                        step,maximum);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
///--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- number of values copied from the iSAR indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to calculate the indicator for two or more bars (it means that the indicator is not yet calculated)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iSARBuffer array is greater than the number of values in the iSAR indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so we copy only new values
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the arrays with values of the iSAR indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iSARBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Parabolic SAR indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iSAR indicator |
//+-----+
bool FillArrayFromBuffer(double &sar_buffer[], // indicator buffer of Parabolic SAR
                        int ind_handle,       // handle of the iSAR indicator
                        int amount            // number of copied values
                        )
{
//--- reset error code

```



```

    ResetLastError();
    //--- fill a part of the iSARBuffer array with values from the indicator buffer that h
    if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iSAR indicator, error code %d",GetLast
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    //--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iRSI

The function returns the handle of the Relative Strength Index indicator. It has only one buffer.

```
int iRSI(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,       // averaging period
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period for the RSI calculation.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iRSI.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRSI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Relative S"

#property indicator_separate_window
```



```

#property indicator_buffers 1
#property indicator_plots 1
//--- drawing iRSI
#property indicator_label1 "iRSI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDodgerBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- limits for displaying of values in the indicator window
#property indicator_maximum 100
#property indicator_minimum 0
//--- horizontal levels in the indicator window
#property indicator_level1 70.0
#property indicator_level2 30.0
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iRSI,          // use iRSI
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iRSI;          // type of the function
input int           ma_period=14;            // period of averaging
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iRSIBuffer[];
//--- variable for storing the handle of the iRSI indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Relative Strength Index indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iRSIBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);

```



```

StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iRSI)
    handle=iRSI(name,period,ma_period,applied_price);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[2];
    //--- period of moving average
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- limit of the step value that can be used for calculations
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_RSI,2,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iRSI indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Relative Strength Index indicator is calculated for
short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),
                        ma_period,applied_price);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],

```



```

        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//--- number of values copied from the iRSI indicator
        int values_to_copy;
//--- determine the number of values calculated in the indicator
        int calculated=BarsCalculated(handle);
        if(calculated<=0)
        {
            PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
            return(0);
        }
//--- if it is the first start of calculation of the indicator or if the number of values calculated is zero
//---or if it is necessary to recalculate the indicator for two or more bars (it means that the indicator is not calculated for a long time)
        if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
        {
            //--- if the iRSIBuffer array is greater than the number of values in the iRSI indicator
            //--- otherwise, we copy less than the size of indicator buffers
            if(calculated>rates_total) values_to_copy=rates_total;
            else
                values_to_copy=calculated;
        }
        else
        {
            //--- it means that it's not the first time of the indicator calculation, and so we copy only new values
            //--- for calculation not more than one bar is added
            values_to_copy=(rates_total-prev_calculated)+1;
        }
//--- fill the array with values of the iRSI indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
        if(!FillArrayFromBuffer(iRSIBuffer,handle,values_to_copy)) return(0);
//--- form the message
        string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                                   TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                   short_name,
                                   values_to_copy);
//--- display the service message on the chart
        Comment(comm);
//--- memorize the number of values in the Relative Strength Index indicator
        bars_calculated=calculated;
//--- return the prev_calculated value for the next call
        return(rates_total);
    }
//+-----+
//| Filling indicator buffers from the iRSI indicator |
//+-----+
bool FillArrayFromBuffer(double &rsi_buffer[], // indicator buffer of Relative Strength Index
                        int ind_handle,        // handle of the iRSI indicator

```



```

        int amount                // number of copied values
    )

    {
    //--- reset error code
        ResetLastError();
    //--- fill a part of the iRSIBuffer array with values from the indicator buffer that h
        if(CopyBuffer(ind_handle,0,0,amount,rsi_buffer)<0)
        {
            //--- if the copying fails, tell the error code
            PrintFormat("Failed to copy data from the iRSI indicator, error code %d",GetLast
            //--- quit with zero result - it means that the indicator is considered as not c
            return(false);
        }
    //--- everything is fine
        return(true);
    }

    //+-----+
    //| Indicator deinitialization function |
    //+-----+
    void OnDeinit(const int reason)
    {
    //--- clear the chart after deleting the indicator
        Comment("");
    }

```


iRVI

The function returns the handle of the Relative Vigor Index indicator.

```
int iRVI(
    string      symbol,      // symbol name
    ENUM_TIMEFRAMES period,  // period
    int         ma_period    // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period for the RVI calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+-----+
//|                                     Demo_iRVI.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRVI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Relative V"

#property indicator_separate_window
#property indicator_buffers 2
```



```

#property indicator_plots 2
//--- the RVI plot
#property indicator_label1 "RVI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- the Signal plot
#property indicator_label2 "Signal"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrRed
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iRVI,          // use iRVI
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iRVI;          // type of the function
input int           ma_period=10;            // period for calculations
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double      RVIBuffer[];
double      SignalBuffer[];
//--- variable for storing the handle of the iRVI indicator
int  handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Relative Vigor Index indicator
int  bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,RVIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);

```



```

StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iRVI)
    handle=iRVI(name,period,ma_period);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    //--- period for calculations
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_RVI,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iRVI indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Relative Vigor Index indicator is calculated for
short_name=StringFormat("iRVI(%s/%s, %d, %d)",name,EnumToString(period),ma_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```



```

{
//--- number of values copied from the iRVI indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values
//---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the RVIBuffer array is greater than the number of values in the iRVI indicator
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the arrays with values of the iRVI indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(RVIBuffer,SignalBuffer,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Relative Vigor Index indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iRVI indicator |
//+-----+
bool FillArrayFromBuffer(double &rvi_buffer[], // indicator buffer of Relative Vigor Index
                        double &signal_buffer[], // indicator buffer of the signal
                        int ind_handle, // handle of the iRVI indicator
                        int amount // number of copied values
                        )
{

```



```

//--- reset error code
ResetLastError();
//--- fill a part of the iRVIBuffer array with values from the indicator buffer that
if(CopyBuffer(ind_handle,0,0,amount,rvi_buffer)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not
    return(false);
}
//--- fill a part of the SignalBuffer array with values from the indicator buffer that
if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLastError());
    //--- quit with zero result - it means that the indicator is considered as not
    return(false);
}
//--- everything is fine
return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iStdDev

The function returns the handle of the Standard Deviation indicator. It has only one buffer.

```
int iStdDev(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    int             ma_shift,         // horizontal shift
    ENUM_MA_METHOD  ma_method,        // smoothing type
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period for the indicator calculations.

ma_shift

[in] Shift of the indicator relative to the price chart.

ma_method

[in] Type of averaging. Can be any of the [ENUM_MA_METHOD](#) values.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iStdDev.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```



```

#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStdDev technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the normal Standard Dev

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iStdDev plot
#property indicator_label1 "iStdDev"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrMediumSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iStdDev,          // use iStdDev
    Call_IndicatorCreate   // use IndicatorCreate
};
//--- input parameters
input Creation          type=Call_iStdDev;          // type of the function
input int               ma_period=20;              // period of averaging
input int               ma_shift=0;                // shift
input ENUM_MA_METHOD    ma_method=MODE_SMA;        // type of smoothing
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string            symbol=" ";                // symbol
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT;    // timeframe
//--- indicator buffer
double          iStdDevBuffer[];
//--- variable for storing the handle of the iStdDev indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Standard Deviation indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iStdDevBuffer,INDICATOR_DATA);

```



```

//--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iStdDev)
        handle=iStdDev(name,period,ma_period,ma_shift,ma_method,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[4];
        //--- period
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        //--- shift
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ma_shift;
        //--- type of smoothing
        pars[2].type=TYPE_INT;
        pars[2].integer_value=ma_method;
        //--- type of price
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_STDDEV,4,pars);
    }
//--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iStdDev indicator for the symbol %s,
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
//--- show the symbol/timeframe the Standard Deviation indicator is calculated for
    short_name=StringFormat("iStdDev(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                                ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price));
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```



```

//--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iStdDev indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iStdDevBuffer array is greater than the number of values in the iStdDev
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the array with values of the Standard Deviation indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iStdDevBuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
}

```



```

//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Standard Deviation indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iStdDev indicator |
//+-----+
bool FillArrayFromBuffer(double &std_buffer[], // indicator buffer of the Standard Deviation
                        int std_shift,         // shift of the Standard Deviation line
                        int ind_handle,        // handle of the iStdDev indicator
                        int amount             // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iStdDevBuffer array with values from the indicator buffer the
    if(CopyBuffer(ind_handle,0,-std_shift,amount,std_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iStdDev indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iStochastic

The function returns the handle of the Stochastic Oscillator indicator.

```
int iStochastic(
    string      symbol,           // symbol name
    ENUM_TIMEFRAMES period,      // period
    int         Kperiod,         // K-period (number of bars for calculations)
    int         Dperiod,         // D-period (period of first smoothing)
    int         slowing,         // final smoothing
    ENUM_MA_METHOD ma_method,    // type of smoothing
    ENUM_STO_PRICE price_field   // stochastic calculation method
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

Kperiod

[in] Averaging period (bars count) for the %K line calculation.

Dperiod

[in] Averaging period (bars count) for the %D line calculation.

slowing

[in] Slowing value.

ma_method

[in] Type of averaging. Can be any of the [ENUM_MA_METHOD](#) values.

price_field

[in] Parameter of price selection for calculations. Can be one of the [ENUM_STO_PRICE](#) values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+-----+
//|                                     Demo_iStochastic.mq5 |
```



```

//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStochastic technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'"
#property description "All the other parameters are similar to the standard Stochastic"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- the Stochastic plot
#property indicator_label1 "Stochastic"
#property indicator_type1  DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Signal plot
#property indicator_label2 "Signal"
#property indicator_type2  DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- set limit of the indicator values
#property indicator_minimum 0
#property indicator_maximum 100
//--- horizontal levels in the indicator window
#property indicator_level1 -100.0
#property indicator_level2 100.0
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iStochastic,    // use iStochastic
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iStochastic; // type of the function
input int           Kperiod=5;             // the K period (the number of K)
input int           Dperiod=3;             // the D period (the period of D)
input int           slowing=3;             // period of final smoothing
input ENUM_MA_METHOD ma_method=MODE_SMA;   // type of smoothing
input ENUM_STO_PRICE price_field=STO_LOWHIGH; // method of calculation of the

```



```

input string          symbol=" ";           // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffers
double              StochasticBuffer[];
double              SignalBuffer[];
//--- variable for storing the handle of the iStochastic indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Stochastic Oscillator indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- assignment of arrays to indicator buffers
    SetIndexBuffer(0,StochasticBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iStochastic)
        handle=iStochastic(name,period,Kperiod,Dperiod,slowing,ma_method,price_field);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[5];
        //--- the K period for calculations
        pars[0].type=TYPE_INT;
        pars[0].integer_value=Kperiod;
        //--- the D period for primary smoothing
        pars[1].type=TYPE_INT;
        pars[1].integer_value=Dperiod;
        //--- the K period for final smoothing
        pars[2].type=TYPE_INT;
        pars[2].integer_value=slowing;
        //--- type of smoothing

```



```

    pars[3].type=TYPE_INT;
    pars[3].integer_value=ma_method;
    //--- method of calculation of the Stochastic
    pars[4].type=TYPE_INT;
    pars[4].integer_value=price_field;
    handle=IndicatorCreate(name,period,IND_STOCHASTIC,5,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iStochastic indicator for the symbol
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Stochastic Oscillator indicator is calculated for
short_name=StringFormat("iStochastic(%s/%s, %d, %d, %d, %s, %s)",name,EnumToString
                        Kperiod,Dperiod,slowing,EnumToString(ma_method),EnumToStrin
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iStochastic indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of va

```



```

//---or if it is necessary to calculated the indicator for two or more bars (it means
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- if the StochasticBuffer array is greater than the number of values in the
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and s
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- fill the arrays with values of the iStochastic indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
if(!FillArraysFromBuffers(StochasticBuffer,SignalBuffer,handle,values_to_copy)) ret
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                        short_name,
                        values_to_copy);

//--- display the service message on the chart
Comment(comm);

//--- memorize the number of values in the Stochastic Oscillator indicator
bars_calculated=calculated;

//--- return the prev_calculated value for the next call
return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iStochastic indicator |
//+-----+

bool FillArraysFromBuffers(double &main_buffer[], // indicator buffer of Stochastic
                        double &signal_buffer[], // indicator buffer of the signal
                        int ind_handle, // handle of the iStochastic indi
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();

    //--- fill a part of the StochasticBuffer array with values from the indicator buffer
    if(CopyBuffer(ind_handle,MAIN_LINE,0,amount,main_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }

    //--- fill a part of the SignalBuffer array with values from the indicator buffer that

```



```
if(CopyBuffer(ind_handle,SIGNAL_LINE,0,amount,signal_buffer)<0)
{
    //--- if the copying fails, tell the error code
    PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
    //--- quit with zero result - it means that the indicator is considered as not c
    return(false);
}
//--- everything is fine
return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iTEMA

The function returns the handle of the Triple Exponential Moving Average indicator. It has only one buffer.

```
int iTEMA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,        // averaging period
    int             ma_shift,         // horizontal shift of indicator
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculation.

ma_shift

[in] Shift of indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iTEMA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```



```

#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Triple Exp

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iTEMA plot
#property indicator_label1 "iTEMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iTEMA,          // use iTEMA
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iTEMA;          // type of the function
input int           ma_period=14;             // period of averaging
input int           ma_shift=0;               // shift
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iTEMABuffer[];
//--- variable for storing the handle of the iTEMA indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Triple Exponential Moving Average indic
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iTEMABuffer,INDICATOR_DATA);
    //--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
    //--- determine the symbol the indicator is drawn for
    name=symbol;

```



```

//--- delete spaces to the right and to the left
StringTrimRight(name);
StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
if(StringLen(name)==0)
{
    //--- take the symbol of the chart the indicator is attached to
    name=_Symbol;
}
//--- create handle of the indicator
if(type==Call_iTEMA)
    handle=iTEMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[3];
    //--- period
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- shift
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- type of price
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_TEMA,3,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iTEMA indicator for the symbol %s/%s",
        name,
        EnumToString(period),
        GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Triple Exponential Moving Average indicator is calculated on
short_name=StringFormat("iTEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
    ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,

```



```

        const int prev_calculated,
        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])

{
//--- number of values copied from the iTEMA indicator
    int values_to_copy;
//--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
//--- if it is the first start of calculation of the indicator or if the number of values
//---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iTEMABuffer array is greater than the number of values in the iTEMA
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- fill the array with values from the Triple Exponential Moving Average indicator
//--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iTEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Triple Exponential Moving Average indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}

```



```

//+-----+
//| Filling indicator buffers from the iTEMA indicator |
//+-----+
bool FillArrayFromBuffer(double &tema_buffer[], // indicator buffer of Triple Exponent
                        int t_shift,           // shift of the line
                        int ind_handle,        // handle of the iTEMA indicator
                        int amount             // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iTEMABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-t_shift,amount,tema_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iTEMA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iTriX

The function returns the handle of the Triple Exponential Moving Averages Oscillator indicator. It has only one buffer.

```
int iTriX(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             ma_period,       // averaging period
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iTriX.mq5 |
//|               Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTriX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
```



```

#property indicator_buffers 1
#property indicator_plots 1
//--- the iTriX plot
#property indicator_label1 "iTriX"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iTriX,          // use iTriX
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iTriX;          // type of the function
input int           ma_period=14;             // period
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iTriXBuffer[];
//--- variable for storing the handle of the iTriX indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Triple Exponential Moving Averages Oscillator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iTriXBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
}

```



```

    }
    ///--- create handle of the indicator
    if(type==Call_iTriX)
        handle=iTriX(name,period,ma_period,applied_price);
    else
    {
        ///--- fill the structure with parameters of the indicator
        MqlParam pars[2];
        ///--- period
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        ///--- type of price
        pars[1].type=TYPE_INT;
        pars[1].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_TRIX,2,pars);
    }
    ///--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        ///--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iTriX indicator for the symbol %s/%s",
                    name,
                    EnumToString(period),
                    GetLastError());
        ///--- the indicator is stopped early
        return(INIT_FAILED);
    }
    ///--- show the symbol/timeframe the Triple Exponential Moving Averages Oscillator is c
    short_name=StringFormat("iTriX(%s/%s, %d, %s)",name,EnumToString(period),
                            ma_period,EnumToString(applied_price));
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
    ///--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ///--- number of values copied from the iTriX indicator

```



```

    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iTriXBuffer array is greater than the number of values in the iTriX
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the array with values from the Triple Exponential Moving Averages Oscillator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iTriXBuffer,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- display the service message on the chart
    Comment(comm);
    //--- memorize the number of values in the Triple Exponential Moving Averages Oscillator
    bars_calculated=calculated;
    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iTriX indicator |
//+-----+

bool FillArrayFromBuffer(double &trix_buffer[], // indicator buffer of values of Triple
                        int ind_handle,         // handle of the iTriX indicator
                        int amount              // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iTriXBuffer array with values from the indicator buffer that

```



```

    if(CopyBuffer(ind_handle,0,0,amount,triX_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iTriX indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not copied
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


iWPR

The function returns the handle of the Larry Williams' Percent Range indicator. It has only one buffer.

```
int iWPR(
    string      symbol,           // symbol name
    ENUM_TIMEFRAMES period,       // period
    int         calc_period       // averaging period
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

calc_period

[in] Period (bars count) for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iWPR.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iWPR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iWPR plot
#property indicator_label1   "iWPR"
#property indicator_type1    DRAW_LINE
#property indicator_color1   clrCyan
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- set limit of the indicator values
#property indicator_minimum -100
#property indicator_maximum 0
//--- horizontal levels in the indicator window
#property indicator_level1 -20.0
#property indicator_level2 -80.0
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iWPR,          // use iWPR
    Call_IndicatorCreate // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iWPR;          // type of the function
input int           calc_period=14;          // period
input string        symbol=" ";              // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // timeframe
//--- indicator buffer
double iWPRBuffer[];
//--- variable for storing the handle of the iWPR indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Larry Williams' Percent Range indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iWPRBuffer,INDICATOR_DATA);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
}

```



```

//--- create handle of the indicator
if(type==Call_iWPR)
    handle=iWPR(name,period,calc_period);
else
{
    //--- fill the structure with parameters of the indicator
    MqlParam pars[1];
    //--- period
    pars[0].type=TYPE_INT;
    pars[0].integer_value=calc_period;
    handle=IndicatorCreate(name,period,IND_WPR,1,pars);
}
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iWPR indicator for the symbol %s/%s,
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Williams' Percent Range indicator is calculated for
short_name=StringFormat("iWPR(%s/%s, %d)",name,EnumToString(period),calc_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iWPR indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {

```



```

        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }

    //--- if it is the first start of calculation of the indicator or if the number of values is less than 2
    //---or if it is necessary to calculate the indicator for two or more bars (it means that the indicator is not calculated yet)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iWPRBuffer array is greater than the number of values in the iWPR indicator buffer
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so we need to copy only new values
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }

    //--- fill the array with values of the Williams' Percent Range indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet, so we return 0
    if(!FillArrayFromBuffer(iWPRBuffer,handle,values_to_copy)) return(0);

    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                              TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                              short_name,
                              values_to_copy);

    //--- display the service message on the chart
    Comment(comm);

    //--- memorize the number of values in the Larry Williams' Percent Range indicator
    bars_calculated=calculated;

    //--- return the prev_calculated value for the next call
    return(rates_total);
}

//+-----+
//| Filling indicator buffers from the iWPR indicator |
//+-----+

bool FillArrayFromBuffer(double &wpr_buffer[], // indicator buffer of Williams' Percent Range
                        int ind_handle,        // handle of the iWPR indicator
                        int amount             // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();

    //--- fill a part of the iWPRBuffer array with values from the indicator buffer that has been calculated
    if(CopyBuffer(ind_handle,0,0,amount,wpr_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iWPR indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated yet
    }
}

```



```
        return(false);
    }
    //--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}
```


iVIDyA

The function returns the handle of the Variable Index Dynamic Average indicator. It has only one buffer.

```
int iVIDyA(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    int             cmo_period,      // period for Chande Momentum
    int             ema_period,      // EMA smoothing period
    int             ma_shift,        // horizontal shift on the price chart
    ENUM_APPLIED_PRICE applied_price // type of price or handle
);
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

cmo_period

[in] Period (bars count) for the Chande Momentum Oscillator calculation.

ema_period

[in] EMA period (bars count) for smoothing factor calculation.

ma_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants [ENUM_APPLIED_PRICE](#) or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iVIDyA.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```



```

#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVIDyA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Variable Index Dynamic

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- the iVIDyA plot
#property indicator_label1 "iVIDyA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iVIDyA,          // use iVIDyA
    Call_IndicatorCreate  // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iVIDyA;          // type of the function
input int           cmo_period=15;             // the Chande Momentum period
input int           ema_period=12;             // period of the factor of smooth
input int           ma_shift=0;                // shift
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // type of price
input string        symbol=" ";               // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double             iVIDyABuffer[];
//--- variable for storing the handle of the iVIDyA indicator
int handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Variable Index Dynamic Average indicator
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iVIDyABuffer,INDICATOR_DATA);

```



```

//--- set shift
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
    name=symbol;
//--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
//--- create handle of the indicator
    if(type==Call_iVIDyA)
        handle=iVIDyA(name,period,cmo_period,ema_period,ma_shift,applied_price);
    else
    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[4];
        //--- the Chande Momentum period
        pars[0].type=TYPE_INT;
        pars[0].integer_value=cmo_period;
        //--- period of the factor of smoothing
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ema_period;
        //--- shift
        pars[2].type=TYPE_INT;
        pars[2].integer_value=ma_shift;
        //--- type of price
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_VIDYA,4,pars);
    }
//--- if the handle is not created
    if(handle==INVALID_HANDLE)
    {
        //--- tell about the failure and output the error code
        PrintFormat("Failed to create handle of the iVIDyA indicator for the symbol %s/%s",
                    name,
                    EnumToString(period),
                    GetLastError());
        //--- the indicator is stopped early
        return(INIT_FAILED);
    }
//--- show the symbol/timeframe the Variable Index Dynamic Average indicator is calculated
    short_name=StringFormat("iVIDyA(%s/%s, %d, %d, %d, %s)",name,EnumToString(period),
                            cmo_period,ema_period,ma_shift,EnumToString(applied_price));
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```



```

//--- normal initialization of the indicator
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    //--- number of values copied from the iVIDyA indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of values
    //---or if it is necessary to calculate the indicator for two or more bars (it means
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- if the iVIDyABuffer array is greater than the number of values in the iVIDyA
        //--- otherwise, we copy less than the size of indicator buffers
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
        //--- it means that it's not the first time of the indicator calculation, and so
        //--- for calculation not more than one bar is added
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- fill the array with values from the Variable Index Dynamic Average indicator
    //--- if FillArrayFromBuffer returns false, it means the information is not ready yet,
    if(!FillArrayFromBuffer(iVIDyABuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- form the message
    string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
}

```



```

//--- display the service message on the chart
    Comment(comm);
//--- memorize the number of values in the Variable Index Dynamic Average indicator
    bars_calculated=calculated;
//--- return the prev_calculated value for the next call
    return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iVIDyA indicator |
//+-----+
bool FillArrayFromBuffer(double &vidya_buffer[],// indicator buffer of Variable Index
                        int v_shift,           // shift of the line
                        int ind_handle,        // handle of the iVIDyA indicator
                        int amount            // number of copied values
                        )
{
//--- reset error code
    ResetLastError();
//--- fill a part of the iVIDyABuffer array with values from the indicator buffer that
    if(CopyBuffer(ind_handle,0,-v_shift,amount,vidya_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iVIDyA indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
//--- everything is fine
    return(true);
}
//+-----+
//| Indicator deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear the chart after deleting the indicator
    Comment("");
}

```


iVolumes

The function returns the handle of the Volumes indicator. It has an only one buffer.

```
int iVolumes(
    string          symbol,          // symbol name
    ENUM_TIMEFRAMES period,          // period
    ENUM_APPLIED_VOLUME applied_volume // volume type for calculation
)
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The [NULL](#) value means the current symbol.

period

[in] The value of the period can be one of the [ENUM_TIMEFRAMES](#) values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be any of the [ENUM_APPLIED_VOLUME](#) values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns [INVALID_HANDLE](#). The computer memory can be freed from an indicator that is no more utilized, using the [IndicatorRelease\(\)](#) function, to which the indicator handle is passed.

Example:

```
//+-----+
//|                                     Demo_iVolumes.mq5 |
//|                                     Copyright 2011, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVolumes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- the iVolumes plot
#property indicator_label1 "iVolumes"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
#property indicator_color1 clrGreen, clrRed
```



```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Enumeration of the methods of handle creation |
//+-----+
enum Creation
{
    Call_iVolumes,          // use iVolumes
    Call_IndicatorCreate    // use IndicatorCreate
};
//--- input parameters
input Creation      type=Call_iVolumes;          // type of the function
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // type of volume
input string        symbol=" ";                  // symbol
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;    // timeframe
//--- indicator buffers
double      iVolumesBuffer[];
double      iVolumesColors[];
//--- variable for storing the handle of the iVolumes indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Volumes indicator
int    bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- assignment of array to indicator buffer
    SetIndexBuffer(0,iVolumesBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iVolumesColors,INDICATOR_COLOR_INDEX);
    //--- determine the symbol the indicator is drawn for
    name=symbol;
    //--- delete spaces to the right and to the left
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- if it results in zero length of the 'name' string
    if(StringLen(name)==0)
    {
        //--- take the symbol of the chart the indicator is attached to
        name=_Symbol;
    }
    //--- create handle of the indicator
    if(type==Call_iVolumes)
        handle=iVolumes(name,period,applied_volume);
    else

```



```

    {
        //--- fill the structure with parameters of the indicator
        MqlParam pars[1];
        //--- type of price
        pars[0].type=TYPE_INT;
        pars[0].integer_value=applied_volume;
        handle=IndicatorCreate(name,period,IND_VOLUMES,1,pars);
    }
//--- if the handle is not created
if(handle==INVALID_HANDLE)
{
    //--- tell about the failure and output the error code
    PrintFormat("Failed to create handle of the iVolumes indicator for the symbol %s",
                name,
                EnumToString(period),
                GetLastError());
    //--- the indicator is stopped early
    return(INIT_FAILED);
}
//--- show the symbol/timeframe the Volumes indicator is calculated for
short_name=StringFormat("iVolumes(%s/%s, %s)",name,EnumToString(period),EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- number of values copied from the iVolumes indicator
    int values_to_copy;
    //--- determine the number of values calculated in the indicator
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastError());
        return(0);
    }
    //--- if it is the first start of calculation of the indicator or if the number of val

```



```

//---or if it is necessary to calculated the indicator for two or more bars (it means
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- if the iVolumesBuffer array is greater than the number of values in the i
    //--- otherwise, we copy less than the size of indicator buffers
    if(calculated>rates_total) values_to_copy=rates_total;
    else
        values_to_copy=calculated;
}
else
{
    //--- it means that it's not the first time of the indicator calculation, and s
    //--- for calculation not more than one bar is added
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- fill the arrays with values of the iVolumes indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
if(!FillArraysFromBuffers(iVolumesBuffer,iVolumesColors,handle,values_to_copy)) ret
//--- form the message
string comm=StringFormat("%s ==> Updated value in the indicator %s: %d",
                        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                        short_name,
                        values_to_copy);
//--- display the service message on the chart
Comment(comm);
//--- memorize the number of values in the Volumes indicator
bars_calculated=calculated;
//--- return the prev_calculated value for the next call
return(rates_total);
}
//+-----+
//| Filling indicator buffers from the iVolumes indicator |
//+-----+
bool FillArraysFromBuffers(double &volume_buffer[], // indicator buffer of Volumes
                        double &color_buffer[], // indicator buffer of colors
                        int ind_handle, // handle of the iVolumes indicator
                        int amount // number of copied values
                        )
{
    //--- reset error code
    ResetLastError();
    //--- fill a part of the iVolumesBuffer array with values from the indicator buffer th
    if(CopyBuffer(ind_handle,0,0,amount,volume_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",Get
        //--- quit with zero result - it means that the indicator is considered as not c
        return(false);
    }
    //--- fill a part of the iVolumesColors array with values from the indicator buffer th

```



```

    if(CopyBuffer(ind_handle,1,0,amount,color_buffer)<0)
    {
        //--- if the copying fails, tell the error code
        PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",GetLastError());
        //--- quit with zero result - it means that the indicator is considered as not calculated
        return(false);
    }
    //--- everything is fine
    return(true);
}

//+-----+
//| Indicator deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //--- clear the chart after deleting the indicator
    Comment("");
}

```


Working with Optimization Results

Functions for organizing custom processing of the optimization results in the strategy tester. They can be called during optimization in testing agents, as well as locally in Expert Advisors and scripts.

When you run an Expert Advisor in the strategy tester, you can create your own data array based on the simple types or [simple structures](#) (they do not contain strings, class objects or objects of dynamic arrays). This data set can be saved using the [FrameAdd\(\)](#) function in a special structure called a frame. During the optimization of an Expert Advisor, each agent can send a series of frames to the terminal. All the received frames are written in the *.MQD file named as the Expert Advisor in the terminal_directory\MQL5\Files\Tester folder. They are written in the order they are received from the agents. Receipt of a frame in the client terminal from a testing agent generates the [TesterPass](#) event.

Frames can be stored in the computer memory and in a file with the specified name. The MQL5 language sets no limitations on the number of frames.

Function	Action
FrameFirst	Moves a pointer of frame reading to the beginning and resets the previously set filter
FrameFilter	Sets the frame reading filter and moves the pointer to the beginning
FrameNext	Reads a frame and moves the pointer to the next one
FrameInputs	Receives input parameters , on which the frame is formed
FrameAdd	Adds a frame with data
ParameterGetRange	Receives data on the values range and the change step for an input variable when optimizing an Expert Advisor in the Strategy Tester
ParameterSetRange	Specifies the use of input variable when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values

See also

[Testing Statistics](#), [Properties of a Running MQL5 Program](#)

FrameFirst

Moves a pointer of frame reading to the beginning and resets a set filter.

```
bool FrameFirst();
```

Return Value

Returns true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

FrameFilter

Sets the frame reading filter and moves the pointer to the beginning.

```
bool FrameFilter(  
    const string name,          // Public name/label  
    long id                    // Public ID  
);
```

Return Value

Returns true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

Note

If an empty string is passed as the first parameter, the filter will work only with a numeric parameter, i.e. only frames with the specified id will be viewed. If the value of the second parameter is [ULONG_MAX](#), only a text filter works.

Call of [FrameFilter\("", ULONG_MAX\)](#) is equivalent to calling [FrameFirst\(\)](#), i.e. equal to not using any filter.

FrameNext

Reads a frame and moves the pointer to the next one. There are two variants of the function.

1. Calling to receive one numeric value

```
bool FrameNext(  
    ulong& pass,      // The number of a pass in the optimization, during which the  
    string& name,      // Public name/label  
    long& id,         // Public ID  
    double& value     // Value  
);
```

2. Calling to receive all the data of a frame

```
bool FrameNext(  
    ulong& pass,      // The number of a pass in the optimization, during which the  
    string& name,      // Public name/label  
    long& id,         // Public ID  
    double& value,    // Value  
    void& data[]      // Array of any type  
);
```

Parameters

pass

[out] The number of a pass during optimization in the strategy tester.

name

[out] The name of the identifier.

id

[out] The value of the identifier.

value

[out] A single numeric value.

data

[out] An array of any type.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

Note

In the second version of the call, you must correctly handle the received data in the *data[]* array.

FrameInputs

Receives [input parameters](#), on which the frame with the specified pass number is formed.

```
bool FrameInputs(  
    ulong    pass,                // The number of a pass in the optimization  
    string&  parameters[],        // An array of strings of form "parameterN=valueN"  
    uint&    parameters_count     // The total number of parameters  
);
```

Parameters

pass

[out] The number of a pass during optimization in the strategy tester.

parameters

[out] A string array with the description of names and parameter values

parameters_count

[out] The number of elements in the array *parameters[]*.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

Note

Having obtained the number of strings *parameters_count* in the *parameters[]* array, you can organize a loop to go through all records. This will help you find the values of input parameters of an Expert Advisor for the specified pass number.

FrameAdd

Adds a frame with data. There are two variants of the function.

1. Adding data from a file

```
bool FrameAdd(  
    const string name,      // Public name/label  
    long id,               // Public ID  
    double value,          // Value  
    const string filename   // Name of a data file  
);
```

2. Adding data from an array of any type

```
bool FrameAdd(  
    const string name,      // Public name/label  
    long id,               // Public ID  
    double value,          // Value  
    const void& data[]      // Array of any type  
);
```

Parameters

name

[in] Public frame label. It can be used for a filter in the [FrameFilter\(\)](#) function.

id

[in] A public identifier of the frame. It can be used for a filter in the [FrameFilter\(\)](#) function.

value

[in] A numeric value to write into the frame. It is used to transmit a single pass result like in the [OnTester\(\)](#) function.

filename

[in] The name of the file that contains data to add to the frame. The file must be located in the folder MQL5/Files.

data

[in] An array of any type to write into the frame. Passed by reference.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the [GetLastError\(\)](#) function.

ParameterGetRange

Receives data on the values range and the change step for an [input variable](#) when optimizing an Expert Advisor in the Strategy Tester. There are 2 variants of the function.

1. Receiving data for the integer type input parameter

```
bool ParameterGetRange(
    const string name,           // parameter (input variable) name
    bool& enable,               // parameter optimization enabled
    long& value,                // parameter value
    long& start,                // initial value
    long& step,                 // change step
    long& stop                  // final value
);
```

2. Receiving data for the real type input parameter

```
bool ParameterGetRange(
    const string name,           // parameter (input variable) name
    double& enable,              // parameter optimization enabled
    double& value,               // parameter value
    double& start,               // initial value
    double& step,                // change step
    double& stop                 // final value
);
```

Parameters

name

[in] [input variable](#) ID. These variables are external parameters of an application. Their values can be specified when launching on a chart or during a single test.

enable

[out] Flag that this parameter can be used to enumerate the values during the optimization in the Strategy Tester.

value

[out] Parameter value.

start

[out] Initial parameter value during the optimization.

step

[out] Parameter change step when enumerating its values.

stop

[out] Final parameter value during the optimization.

Return Value

Returns true if successful, otherwise false. For information about the error, use the [GetLastError\(\)](#) function.

Note

The function can be called only from [OnTesterInit\(\)](#), [OnTesterPass\(\)](#) and [OnTesterDeinit\(\)](#) handlers. It has been introduced to receive Expert Advisor input parameters' values and variation ranges during the optimization in the Strategy Tester.

When called in [OnTesterInit\(\)](#), the obtained data can be used to redefine the rules for enumeration of any [input variable](#) using [ParameterSetRange\(\)](#) function. Therefore, new Start, Stop and Step values can be set and the input parameter can even be completely excluded from optimization regardless of the Strategy Tester settings. This allows you to manage the area of the input parameters during the optimization by excluding some parameters from the optimization depending on the Expert Advisor's key parameters' values.

Example:

```
#property description "Expert Advisor for ParameterGetRange() function demonstration."
#property description "Should be launched in the optimization mode of the Strategy Tester"

//--- input parameters
input int          Input1=1;
input double       Input2=2.0;
input bool         Input3=false;
input ENUM_DAY_OF_WEEK Input4=SUNDAY;

//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
    //--- Expert Advisor is designed for operation only in the Strategy Tester
    if(!MQL5InfoInteger(MQL5_OPTIMIZATION))
    {
        MessageBox("Should be launched in the optimization mode of the Strategy Tester!");
        //--- finish the Expert Advisor operation in advance and remove from the chart
        return(INIT_FAILED);
    }

    //--- successful completion of initialization
    return(INIT_SUCCEEDED);
}

//+-----+
//| TesterInit function |
//+-----+

void OnTesterInit()
{
    //--- example for long type input parameter
    string name="Input1";
    bool enable;
    long par1,par1_start,par1_step,par1_stop;
    ParameterGetRange(name,enable,par1,par1_start,par1_step,par1_stop);
}
```



```

Print("First parameter");
PrintFormat("%s=%d  enable=%s  from %d to %d with step=%d",
            name,par1,(string)enable,par1_start,par1_stop,par1_step);
//--- example for double type input parameter
name="Input2";
double par2,par2_start,par2_step,par2_stop;
ParameterGetRange(name,enable,par2,par2_start,par2_step,par2_stop);
Print("Second parameter");
PrintFormat("%s=%G  enable=%s  from %G to %G with step=%G",
            name,par2,(string)enable,par2_start,par2_stop,par2_step);

//--- example for bool type input parameter
name="Input3";
long par3,par3_start,par3_step,par3_stop;
ParameterGetRange(name,enable,par3,par3_start,par3_step,par3_stop);
Print("Third parameter");
PrintFormat("%s=%s  enable=%s  from %s to %s",
            name,(string)par3,(string)enable,
            (string)par3_start,(string)par3_stop);

//--- example for enumeration type input parameter
name="Input4";
long par4,par4_start,par4_step,par4_stop;
ParameterGetRange(name,enable,par4,par4_start,par4_step,par4_stop);
Print("Fourth parameter");
PrintFormat("%s=%s  enable=%s  from %s to %s",
            name,EnumToString((ENUM_DAY_OF_WEEK)par4),(string)enable,
            EnumToString((ENUM_DAY_OF_WEEK)par4_start),
            EnumToString((ENUM_DAY_OF_WEEK)par4_stop));
}
//+-----+
//| TesterDeinit function |
//+-----+
void OnTesterDeinit()
{
//--- this message will be shown after optimization is complete
Print(__FUNCTION__," Optimization completed");
}

```


ParameterSetRange

Specifies the use of [input variable](#) when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values. There are 2 variants of the function.

1. Specifying the values for the integer type input parameter

```
bool ParameterSetRange(
    const string name,          // parameter (input variable) name
    bool enable,               // parameter optimization enabled
    long value,                // parameter value
    long start,                // initial value
    long step,                 // change step
    long stop                  // final value
);
```

2. Specifying the values for the real type input parameter

```
bool ParameterSetRange(
    const string name,          // parameter (input variable) name
    double enable,              // parameter optimization enabled
    double value,               // parameter value
    double start,               // initial value
    double step,                // change step
    double stop                 // final value
);
```

Parameters

name

[in] [input or sinput](#) variable ID. These variables are external parameters of an application. Their values can be specified when launching the program.

enable

[in] Enable this parameter to enumerate the values during the optimization in the Strategy Tester.

value

[in] Parameter value.

start

[in] Initial parameter value during the optimization.

step

[in] Parameter change step when enumerating its values.

stop

[in] Final parameter value during the optimization.

Return Value

Returns true if successful, otherwise false. For information about the error, use the [GetLastError\(\)](#) function.

Note

The function can be called only from [OnTesterInit\(\)](#) handler when launching optimization from the Strategy Tester. It is designed for specifying the parameter's range and change step. The parameter can be completely excluded from optimization regardless of the Strategy Tester settings. It also allows using the variables declared with sinput modifier in the optimization process.

ParameterSetRange() function allows you to manage an Expert Advisor optimization in the Strategy Tester depending on its key parameters' values by including or excluding required input parameters from the optimization and setting the required range and the change step.

Event Functions

This group contains functions for working with custom events and timer events. Besides this group, there are special functions for handling [predefined events](#).

Function	Action
EventSetMillisecondTimer	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart
EventSetTimer	Starts the timer event generator with the specified periodicity for the current chart
EventKillTimer	Stops the generation of events by the timer in the current chart
EventChartCustom	Generates a custom event for the specified chart

See also

[Types of Chart Events](#)

EventSetMillisecondTimer

The function indicates to the client terminal that [timer](#) events should be generated at intervals less than one second for this Expert Advisor or indicator.

```
bool EventSetMillisecondTimer(  
    int milliseconds // number of milliseconds  
);
```

Parameters

milliseconds

[in] Number of milliseconds defining the frequency of timer events.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an [error](#) code, [GetLastError\(\)](#) function should be called.

Note

This feature is designed for the cases when high-resolution timer is required. In other words, timer events should be received more frequently than once per second. If a conventional timer with the period of more than one second is enough for you, use [EventSetTimer\(\)](#).

The minimum interval of 1000 milliseconds is used in the strategy tester. In general, when the timer period is reduced, the testing time is increased, as the handler of timer events is called more often. When working in real-time mode, timer events are generated no more than 1 time in 10-16 milliseconds due to hardware limitations.

Usually, this function should be called from [OnInit\(\)](#) function or in class [constructor](#). To handle events coming from the timer, an Expert Advisor or an indicator should have [OnTimer\(\)](#) function.

Each Expert Advisor and each indicator work with its own timer receiving events solely from this timer. During mql5 application shutdown, the timer is forcibly destroyed in case it has been created but has not been disabled by [EventKillTimer\(\)](#) function.

Only one timer can be launched for each program. Each mql5 application and chart have their own queue of events where all newly arrived events are placed. If the queue already contains [Timer](#) event or this event is in the processing stage, then the new Timer event is not added to mql5 application queue.

EventSetTimer

The function indicates to the client terminal, that for this indicator or Expert Advisor, events from the [timer](#) must be generated with the specified periodicity.

```
bool EventSetTimer(  
    int seconds // number of seconds  
);
```

Parameters

seconds

[in] Number of seconds that determine the frequency of the timer event occurrence.

Return Value

In case of success returns true, otherwise false. In order to get an [error code](#), the [GetLastError\(\)](#) function should be called.

Note

Normally, this function must be called from the [OnInit\(\)](#) function or from a class [constructor](#). In order to handle events coming from the timer, the Expert Advisor must have the [OnTimer\(\)](#) function.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql5 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the [EventKillTimer\(\)](#) function.

For each program no more than one timer can be run. Each mql5 program and each chart has its own queue of events, in which all the newly received events are placed. If the [Timer](#) event is present in the queue or is being processed, the new Timer event will not be placed in the queue of the mql5 program.

EventKillTimer

Specifies the client terminal that is necessary to stop the generation of events from [Timer](#).

```
void EventKillTimer();
```

Return Value

No return value.

Note

Typically, this function must be called from a function [OnDeinit\(\)](#), if the [EventSetTimer\(\)](#) function has been called from [OnInit\(\)](#). This function can also be called from the class destructor, if the [EventSetTimer\(\)](#) function has been called in the [constructor](#) of this class.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql5 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the [EventKillTimer\(\)](#) function

EventChartCustom

The function generates a custom event for the specified chart.

```
bool EventChartCustom(
    long    chart_id,           // identifier of the event receiving chart
    ushort  custom_event_id,    // event identifier
    long    lparam,            // parameter of type long
    double  dparam,            // parameter of type double
    string  sparam             // string parameter of the event
);
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

custom_event_id

[in] ID of the user events. This identifier is automatically added to the value [CHARTEVENT_CUSTOM](#) and converted to the integer type.

lparam

[in] Event parameter of the long type passed to the [OnChartEvent](#) function.

dparam

[in] Event parameter of the double type passed to the [OnChartEvent](#) function.

sparam

[in] Event parameter of the string type passed to the [OnChartEvent](#) function. If the string is longer than 63 characters, it is truncated.

Return Value

Returns true if a custom event has been successfully placed in the events queue of the chart that receives the events. In case of failure, it returns false. Use [GetLastError\(\)](#) to get an error code.

Note

An Expert Advisor or indicator attached to the specified chart handles the event using the function [OnChartEvent](#)(int event_id, long& lparam, double& dparam, string& sparam).

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

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of times the keystroke is	The string value of a bit mask describing the

			repeated as a result of the user holding down the key)	status of keyboard buttons
Mouse event (if property CHART_EVENT_MOUSE_MOVE =true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
Event of graphical object creation (if CHART_EVENT_OBJECT_CREATE =true is set for the chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE =true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text editing in the entry box of the	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit graphical object, in which text

LabelEdit graphical object				editing has completed
Event of changes on a chart	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

Example:

```
//+-----+
//|                                     ButtonClickExpert.mq5 |
//|                                     Copyright 2009, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;

//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
    //--- Create a button to send custom events
    ObjectCreate(0,buttonID,OBJ_BUTTON,0,100,100);
    ObjectSetInteger(0,buttonID,OBJPROP_COLOR,clrWhite);
    ObjectSetInteger(0,buttonID,OBJPROP_BGCOLOR,clrGray);
    ObjectSetInteger(0,buttonID,OBJPROP_XDISTANCE,100);
    ObjectSetInteger(0,buttonID,OBJPROP_YDISTANCE,100);
    ObjectSetInteger(0,buttonID,OBJPROP_XSIZE,200);
    ObjectSetInteger(0,buttonID,OBJPROP_YSIZE,50);
    ObjectSetString(0,buttonID,OBJPROP_FONT,"Arial");
    ObjectSetString(0,buttonID,OBJPROP_TEXT,"Button");
    ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
    ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

    //--- Create a label for displaying information
    ObjectCreate(0,labelID,OBJ_LABEL,0,100,100);
    ObjectSetInteger(0,labelID,OBJPROP_COLOR,clrRed);
    ObjectSetInteger(0,labelID,OBJPROP_XDISTANCE,100);
    ObjectSetInteger(0,labelID,OBJPROP_YDISTANCE,50);
    ObjectSetString(0,labelID,OBJPROP_FONT,"Trebuchet MS");
    ObjectSetString(0,labelID,OBJPROP_TEXT,"No information");
```



```

ObjectSetInteger(0,labelID,OBJPROP_FONTSIZE,20);
ObjectSetInteger(0,labelID,OBJPROP_SELECTABLE,0);

//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
ObjectDelete(0,buttonID);
ObjectDelete(0,labelID);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//---

}
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- Check the event by pressing a mouse button
if(id==CHARTEVENT_OBJECT_CLICK)
{
string clickedChartObject=sparam;
//--- If you click on the object with the name buttonID
if(clickedChartObject==buttonID)
{
//--- State of the button - pressed or not
bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
//--- log a debug message
Print("Button pressed = ",selected);
int customEventID; // Number of the custom event to send
string message;    // Message to be sent in the event
//--- If the button is pressed
if(selected)
{
message="Button pressed";
customEventID=CHARTEVENT_CUSTOM+1;
}
else // Button is not pressed

```



```

        {
            message="Button in not pressed";
            customEventID=CHARTEVENT_CUSTOM+999;
        }
        //--- Send a custom event "our" chart
        EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
        ///--- Send a message to all open charts
        BroadcastEvent(ChartID(),0,"Broadcast Message");
        //--- Debug message
        Print("Sent an event with ID = ",customEventID);
    }
    ChartRedraw();// Forced redraw all chart objects
}

//--- Check the event belongs to the user events
if(id>CHARTEVENT_CUSTOM)
{
    if(id==broadcastEventID)
    {
        Print("Got broadcast message from a chart with id = "+lparam);
    }
    else
    {
        //--- We read a text message in the event
        string info=sparam;
        Print("Handle the user event with the ID = ",id);
        //--- Display a message in a label
        ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
        ChartRedraw();// Forced redraw all chart objects
    }
}

}

//+-----+
//| sends broadcast event to all open charts |
//+-----+
void BroadcastEvent(long lparam,double dparam,string sparam)
{
    int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
    long currChart=ChartFirst();
    int i=0;
    while(i<CHARTS_MAX) // We have certainly no more than CHARTS_MAX of
    {
        EventChartCustom(currChart,eventID,lparam,dparam,sparam);
        currChart=ChartNext(currChart); // We have received a new chart from the previous
        if(currChart==-1) break; // Reached the end of the charts list
        i++; // Do not forget to increase the counter
    }
}

//+-----+

```


See also

[Events of the client terminal](#), [Event handler functions](#)

Working with OpenCL

[OpenCL](#) programs are used for performing computations on video cards that support OpenCL 1.1 or higher. Modern video cards contain hundreds of small specialized processors that can simultaneously perform simple mathematical operations with incoming data streams. The OpenCL language organizes parallel computing and provides greater speed for a certain class of tasks.

In some graphic cards working with the [double](#) type numbers is disabled by default. This can lead to compilation error 5105. To enable support for the double type numbers, please add the following directive to your OpenCL program: [#pragma OPENCL EXTENSION cl_khr_fp64 : enable](#). However if a graphic card doesn't support double, enabling this directive won't be of help.

It is recommended to write the source code for OpenCL in separate CL files, which can later be included in the MQL5 program using the [resource variables](#).

Functions for running programs in OpenCL:

Function	Action
CLHandleType	Returns the type of an OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration
CLGetInfoInteger	Returns the value of an integer property for an OpenCL object or device
CLContextCreate	Creates an OpenCL context
CLContextFree	Removes an OpenCL context
CLGetDeviceInfo	Receives device property from OpenCL driver
CLProgramCreate	Creates an OpenCL program from a source code
CLProgramFree	Removes an OpenCL program
CLKernelCreate	Creates an OpenCL start function
CLKernelFree	Removes an OpenCL start function
CLSetKernelArg	Sets a parameter for the OpenCL function
CLSetKernelArgMem	Sets an OpenCL buffer as a parameter of the OpenCL function
CLSetKernelArgMemLocal	Sets the local buffer as an argument of the kernel function
CLBufferCreate	Creates an OpenCL buffer
CLBufferFree	Deletes an OpenCL buffer
CLBufferWrite	Writes an array into an OpenCL buffer
CLBufferRead	Reads an OpenCL buffer into an array
CLExecute	Runs an OpenCL program

<u>CLExecutionStatus</u>	Returns the OpenCL program execution status
--	---

See also[OpenCL](#), [Resources](#)

CLHandleType

Returns the type of an OpenCL handle as a value of the `ENUM_OPENCL_HANDLE_TYPE` enumeration.

```
ENUM_OPENCL_HANDLE_TYPE CLHandleType(  
    int handle    // Handle of an OpenCL object  
);
```

Parameters

handle

[in] A handle to an OpenCL object: a context, a kernel or an OpenCL program.

Return Value

The type of the OpenCL handle as a value of the [ENUM_OPENCL_HANDLE_TYPE](#) enumeration.

ENUM_OPENCL_HANDLE_TYPE

Identifier	Description
OPENCL_INVALID	Incorrect handle
OPENCL_CONTEXT	A handle of the OpenCL context
OPENCL_PROGRAM	A handle of the OpenCL program
OPENCL_KERNEL	A handle of the OpenCL kernel
OPENCL_BUFFER	A handle of the OpenCL buffer

CLGetInfoInteger

Returns the value of an integer property for an OpenCL object or device.

```
long CLGetInfoInteger(
    int handle,                // The handle of the OpenCL object or the number of the OpenCL device
    ENUM_OPENCL_PROPERTY_INTEGER prop // Requested property
);
```

Parameters

handle

[in] A handle to the OpenCL object or number of the OpenCL device. Numbering of OpenCL devices starts with zero.

prop

[in] The type of a requested property from the [ENUM_OPENCL_PROPERTY_INTEGER](#) enumeration, the value of which you want to obtain.

Return Value

The value of the property if successful or -1 in case of an error. For information about the error, use the [GetLastError\(\)](#) function.

ENUM_OPENCL_PROPERTY_INTEGER

Identifier	Description	Type
CL_DEVICE_COUNT	The number of devices with OpenCL support. This property does not require specification of the first parameter, i.e. you can pass a zero value for the <i>handle</i> parameter.	int
CL_DEVICE_TYPE	Type of device	ENUM_CL_DEVICE_TYPE
CL_DEVICE_VENDOR_ID	Unique vendor identifier	uint
CL_DEVICE_MAX_COMPUTE_UNITS	Number of parallel calculated tasks in OpenCL device. One working group solves one computational task. The lowest value is 1	uint
CL_DEVICE_MAX_CLOCK_FREQUENCY	Highest set frequency of the device in MHz.	uint
CL_DEVICE_GLOBAL_MEM_SIZE	Size of the global memory of the device in bytes	ulong
CL_DEVICE_LOCAL_MEM_SIZE	Size of the processed data (scene) local memory in bytes	uint

CL_BUFFER_SIZE	Actual size of the OpenCL buffer in bytes	ulong
CL_DEVICE_MAX_WORK_GROUP_SIZE	The total number of the local working groups available for an OpenCL device.	ulong
CL_KERNEL_WORK_GROUP_SIZE	The total number of the local working groups available for an OpenCL program.	ulong
CL_KERNEL_LOCAL_MEM_SIZE	Size of the local memory (in bytes) used by an OpenCL program for solving all parallel tasks in a group. Use CL_DEVICE_LOCAL_MEM_SIZE to receive the maximum available value	ulong
CL_KERNEL_PRIVATE_MEM_SIZE	The minimum size of the private memory (in bytes) used by each task in the OpenCL program kernel.	ulong

The `ENUM_CL_DEVICE_TYPE` enumeration contains possible types of devices supporting OpenCL. You can receive the type of device by its number or the handle of the OpenCL object by calling `CLGetInfoInteger(handle_or_deviceN, CL_DEVICE_TYPE)`.

ENUM_CL_DEVICE_TYPE

Identifier	Description
CL_DEVICE_ACCELERATOR	Dedicated OpenCL accelerators (for example, the IBM CELL Blade). These devices communicate with the host processor using a peripheral interconnect such as PCIe.
CL_DEVICE_CPU	An OpenCL device that is the host processor. The host processor runs the OpenCL implementations and is a single or multi-core CPU.
CL_DEVICE_GPU	An OpenCL device that is a GPU.
CL_DEVICE_DEFAULT	The default OpenCL device in the system. The default device cannot be a <code>CL_DEVICE_TYPE_CUSTOM</code> device.
CL_DEVICE_CUSTOM	Dedicated accelerators that do not support programs written in OpenCL C.

Example:

```
void OnStart()
```



```
{
    int cl_ctx;
    //--- initialize OpenCL context
    if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
    {
        Print("OpenCL not found");
        return;
    }
    //--- Display general information about OpenCL device
    Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_DEVICE_TYPE)));
    Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
    Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
    Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY)," MHz");
    Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE)," bytes");
    Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE)," bytes");
    //---
}
```


CLGetInfoString

Returns string value of a property for OpenCL object or device.

```
bool CLGetInfoString(  
    int handle, // OpenCL object handle or OpenCL device number  
    ENUM_OPENCL_PROPERTY_STRING prop, // requested property  
    string& value // referenced string  
);
```

Parameters

- handle*
- [in] OpenCL object handle or OpenCL device number. The numbering of OpenCL devices starts with zero.
- prop*
- [in] Type of requested property from [ENUM_OPENCL_PROPERTY_STRING](#) enumeration, the value of which should be obtained.
- value*
- [out] String for receiving the property value.

Return Value

true if successful, otherwise false. For information about the error, use the [GetLastError\(\)](#) function.

ENUM_OPENCL_PROPERTY_STRING

Identifier	Description
CL_PLATFORM_PROFILE	CL_PLATFORM_PROFILE - OpenCL Profile. Profile name may be one of the following values: <ul style="list-style-type: none">FULL_PROFILE - implementation supports OpenCL (functionality is defined as the part of the kernel specification without requiring additional extensions for OpenCL support);EMBEDDED_PROFILE - implementation supports OpenCL as a supplement. Amended profile is defined as a subset for each OpenCL version.
CL_PLATFORM_VERSION	OpenCL version
CL_PLATFORM_VENDOR	Platform vendor name
CL_PLATFORM_EXTENSIONS	List of extensions supported by the platform. Extension names should be supported by all devices related to this platform
CL_DEVICE_NAME	Device name
CL_DEVICE_VENDOR	Vendor name

CL_DRIVER_VERSION	OpenCL driver version in major_number.minor_number format
CL_DEVICE_PROFILE	OpenCL device profile. Profile name may be one of the following values: <ul style="list-style-type: none"> • FULL_PROFILE - implementation supports OpenCL (functionality is defined as the part of the kernel specification without requiring additional extensions for OpenCL support); • EMBEDDED_PROFILE - implementation supports OpenCL as a supplement. Amended profile is defined as a subset for each OpenCL version.
CL_DEVICE_VERSION	OpenCL version in "OpenCL<space><major_version.minor_version><space><vendor-specific information>" format
CL_DEVICE_EXTENSIONS	List of extensions supported by the device. The list may contain extensions supported by the vendor, as well as one or more approved names: <ul style="list-style-type: none"> cl_khr_int64_base_atomics cl_khr_int64_extended_atomics cl_khr_fp16 cl_khr_gl_sharing cl_khr_gl_event cl_khr_d3d10_sharing cl_khr_dx9_media_sharing cl_khr_d3d11_sharing
CL_DEVICE_BUILT_IN_KERNELS	The list of supported kernels separated by ";".
CL_DEVICE_OPENCL_C_VERSION	The maximum version supported by the compiler for this device. Version format: "OpenCL<space>C<space><major_version.minor_version><space><vendor-specific information> "

Example:

```

void OnStart()
{
    int cl_ctx;
    string str;
    //--- initialize OpenCL context
    if ((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY)) == INVALID_HANDLE)
    {
        Print("OpenCL not found");
        return;
    }
    //--- Display information about the platform
    if (CLGetInfoString(cl_ctx, CL_PLATFORM_NAME, str))
        Print("OpenCL platform name: ", str);
}

```



```

    if (CLGetInfoString(cl_ctx, CL_PLATFORM_VENDOR, str))
        Print("OpenCL platform vendor: ", str);
    if (CLGetInfoString(cl_ctx, CL_PLATFORM_VERSION, str))
        Print("OpenCL platform ver: ", str);
    if (CLGetInfoString(cl_ctx, CL_PLATFORM_PROFILE, str))
        Print("OpenCL platform profile: ", str);
    if (CLGetInfoString(cl_ctx, CL_PLATFORM_EXTENSIONS, str))
        Print("OpenCL platform ext: ", str);
//--- Display information about the device
    if (CLGetInfoString(cl_ctx, CL_DEVICE_NAME, str))
        Print("OpenCL device name: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_PROFILE, str))
        Print("OpenCL device profile: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_BUILT_IN_KERNELS, str))
        Print("OpenCL device kernels: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_EXTENSIONS, str))
        Print("OpenCL device ext: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_VENDOR, str))
        Print("OpenCL device vendor: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_VERSION, str))
        Print("OpenCL device ver: ", str);
    if (CLGetInfoString(cl_ctx, CL_DEVICE_OPENCL_C_VERSION, str))
        Print("OpenCL open c ver: ", str);
//--- Display general information about the OpenCL device
    Print("OpenCL type: ", EnumToString((ENUM_CL_DEVICE_TYPE) CLGetInfoInteger(cl_ctx, CL_
    Print("OpenCL vendor ID: ", CLGetInfoInteger(cl_ctx, CL_DEVICE_VENDOR_ID));
    Print("OpenCL units: ", CLGetInfoInteger(cl_ctx, CL_DEVICE_MAX_COMPUTE_UNITS));
    Print("OpenCL freq: ", CLGetInfoInteger(cl_ctx, CL_DEVICE_MAX_CLOCK_FREQUENCY));
    Print("OpenCL global mem: ", CLGetInfoInteger(cl_ctx, CL_DEVICE_GLOBAL_MEM_SIZE));
    Print("OpenCL local mem: ", CLGetInfoInteger(cl_ctx, CL_DEVICE_LOCAL_MEM_SIZE));
//---
}

```


CLContextCreate

Creates an OpenCL context and returns its handle.

```
int CLContextCreate(  
    int device    // Serial number of the OpenCL device or macro  
);
```

Parameter

device

[in] The ordinal number of the OpenCL-device in the system. Instead of a specific number, you can specify one of the following values:

- CL_USE_ANY - any available device with OpenCL support is allowed;
- CL_USE_CPU_ONLY - only OpenCL emulation on CPU is allowed;
- CL_USE_GPU_ONLY - OpenCL emulation is prohibited and only specialized devices with OpenCL support (video cards) can be used.

Return Value

A handle to the OpenCL context if successful, otherwise -1. For information about the error, use the [GetLastError\(\)](#) function.

CLContextFree

Removes an OpenCL context.

```
void CLContextFree(  
    int context    // Handle to an OpenCL context  
);
```

Parameters

context

[in] Handle of the OpenCL context.

Return Value

None. In the case of an internal error the value of [_LastError](#) changes. For information about the error, use the [GetLastError\(\)](#) function.

CLGetDeviceInfo

The function receives device property from OpenCL driver.

```
bool CLGetDeviceInfo(  
    int      handle,           // OpenCL device handle  
    int      property_id,      // requested property ID  
    uchar&   data[],          // array for receiving data  
    uint&    size              // shift in the array elements, default value is 0  
);
```

Parameters

handle

[in] OpenCL device index or OpenCL handle created by [CLContextCreate\(\)](#) function.

property_id

[in] ID of the OpenCL device property that should be received. The values can be of one of the predetermined ones listed in the [table below](#).

data[]

[out] The array for receiving data on the requested property.

size

[out] Size of the received data in the array *data[]*.

Return Value

true if successful, otherwise false. For information about the error, use the [GetLastError\(\)](#) function.

Note

For one-dimensional arrays, the number of the element, from which data reading for OpenCL buffer starts, is calculated considering [AS_SERIES](#) flag.

The list of available IDs of OpenCL device properties

Exact description of the property and its functions can be found at [the official OpenCL web site](#).

Identifier	Value
CL_DEVICE_TYPE	0x1000
CL_DEVICE_VENDOR_ID	0x1001
CL_DEVICE_MAX_COMPUTE_UNITS	0x1002
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	0x1003
CL_DEVICE_MAX_WORK_GROUP_SIZE	0x1004
CL_DEVICE_MAX_WORK_ITEM_SIZES	0x1005
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR	0x1006

CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT	0x1007
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT	0x1008
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG	0x1009
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT	0x100A
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE	0x100B
CL_DEVICE_MAX_CLOCK_FREQUENCY	0x100C
CL_DEVICE_ADDRESS_BITS	0x100D
CL_DEVICE_MAX_READ_IMAGE_ARGS	0x100E
CL_DEVICE_MAX_WRITE_IMAGE_ARGS	0x100F
CL_DEVICE_MAX_MEM_ALLOC_SIZE	0x1010
CL_DEVICE_IMAGE2D_MAX_WIDTH	0x1011
CL_DEVICE_IMAGE2D_MAX_HEIGHT	0x1012
CL_DEVICE_IMAGE3D_MAX_WIDTH	0x1013
CL_DEVICE_IMAGE3D_MAX_HEIGHT	0x1014
CL_DEVICE_IMAGE3D_MAX_DEPTH	0x1015
CL_DEVICE_IMAGE_SUPPORT	0x1016
CL_DEVICE_MAX_PARAMETER_SIZE	0x1017
CL_DEVICE_MAX_SAMPLERS	0x1018
CL_DEVICE_MEM_BASE_ADDR_ALIGN	0x1019
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE	0x101A
CL_DEVICE_SINGLE_FP_CONFIG	0x101B
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE	0x101C
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE	0x101D
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE	0x101E
CL_DEVICE_GLOBAL_MEM_SIZE	0x101F
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE	0x1020
CL_DEVICE_MAX_CONSTANT_ARGS	0x1021
CL_DEVICE_LOCAL_MEM_TYPE	0x1022
CL_DEVICE_LOCAL_MEM_SIZE	0x1023

CL_DEVICE_ERROR_CORRECTION_SUPPORT	0x1024
CL_DEVICE_PROFILING_TIMER_RESOLUTION	0x1025
CL_DEVICE_ENDIAN_LITTLE	0x1026
CL_DEVICE_AVAILABLE	0x1027
CL_DEVICE_COMPILER_AVAILABLE	0x1028
CL_DEVICE_EXECUTION_CAPABILITIES	0x1029
CL_DEVICE_QUEUE_PROPERTIES	0x102A
CL_DEVICE_NAME	0x102B
CL_DEVICE_VENDOR	0x102C
CL_DRIVER_VERSION	0x102D
CL_DEVICE_PROFILE	0x102E
CL_DEVICE_VERSION	0x102F
CL_DEVICE_EXTENSIONS	0x1030
CL_DEVICE_PLATFORM	0x1031
CL_DEVICE_DOUBLE_FP_CONFIG	0x1032
CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF	0x1034
CL_DEVICE_HOST_UNIFIED_MEMORY	0x1035
CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR	0x1036
CL_DEVICE_NATIVE_VECTOR_WIDTH_SHORT	0x1037
CL_DEVICE_NATIVE_VECTOR_WIDTH_INT	0x1038
CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG	0x1039
CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT	0x103A
CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE	0x103B
CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF	0x103C
CL_DEVICE_OPENCL_C_VERSION	0x103D
CL_DEVICE_LINKER_AVAILABLE	0x103E
CL_DEVICE_BUILT_IN_KERNELS	0x103F
CL_DEVICE_IMAGE_MAX_BUFFER_SIZE	0x1040
CL_DEVICE_IMAGE_MAX_ARRAY_SIZE	0x1041
CL_DEVICE_PARENT_DEVICE	0x1042
CL_DEVICE_PARTITION_MAX_SUB_DEVICES	0x1043

CL_DEVICE_PARTITION_PROPERTIES	0x1044
CL_DEVICE_PARTITION_AFFINITY_DOMAIN	0x1045
CL_DEVICE_PARTITION_TYPE	0x1046
CL_DEVICE_REFERENCE_COUNT	0x1047
CL_DEVICE_PREFERRED_INTEROP_USER_SYNC	0x1048
CL_DEVICE_PRINTF_BUFFER_SIZE	0x1049
CL_DEVICE_IMAGE_PITCH_ALIGNMENT	0x104A
CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMEN T	0x104B

Example:

```

void OnStart()
{
//---
    int dCount= CLGetInfoInteger(0,CL_DEVICE_COUNT);
    for(int i = 0; i<dCount; i++)
    {
        int clCtx=CLContextCreate(i);
        if(clCtx == -1)
            Print("ERROR in CLContextCreate");
        string device;
        CLGetInfoString(clCtx,CL_DEVICE_NAME,device);
        Print(i," : ",device);
        uchar data[1024];
        uint size;
        CLGetDeviceInfo(clCtx,CL_DEVICE_VENDOR,data,size);
        Print("size = ",size);
        string str=CharArrayToString(data);
        Print(str);
    }
}

//--- example of entries in Experts journal
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      2: Advanced Micro Devices, Inc.
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      size = 32
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      Tahiti
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      Intel(R) Corporation
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      size = 21
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      1:      Intel(R) Core(TM) i7-37
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      NVIDIA Corporation
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      size = 19
// 2013.07.24 10:50:48      openc1 (EURUSD,H1)      0: GeForce GTX 580

```


CLProgramCreate

Creates an OpenCL program from a source code.

```
int CLProgramCreate(
    int          context,    // Handle to an OpenCL context
    const string source      // Source code
);
```

An overloaded function version creates an OpenCL program and writes compiler messages into the passed string.

```
int CLProgramCreate(
    int          context,    // Handle to an OpenCL context
    const string source,    // Source code
    string       &build_log // A string for receiving the compilation log
);
```

Parameters

context

[in] Handle of the OpenCL context.

source

[in] String with the source code of the OpenCL program.

&build_log

[in] A string for receiving the OpenCL compiler messages.

Return Value

A handle to an OpenCL object if successful. In case of error -1 is returned. For information about the error, use the [GetLastError\(\)](#) function.

Note

At the moment, the following error codes are used:

- `ERR_OPENCL_INVALID_HANDLE` - invalid handle to the context OpenCL.
- `ERR_INVALID_PARAMETER` - invalid string parameter.
- `ERR_NOT_ENOUGH_MEMORY` - not enough memory to complete operation.
- `ERR_OPENCL_PROGRAM_CREATE` - internal error of OpenCL or compilation error.

In some graphic cards working with the [double](#) type numbers is disabled by default. This can lead to compilation error 5105. To enable support for the double type numbers, please add the following directive to your OpenCL program: `#pragma OPENCL EXTENSION cl_khr_fp64 : enable`. However if a graphic card doesn't support double, enabling this directive won't be of help.

Example:

```
//+-----+
//| OpenCL kernel |
//+-----+
const string
```



```

cl_src=
    //--- by default some GPU doesn't support doubles
    //--- cl_khr_fp64 directive is used to enable work with doubles
    "#pragma OPENCL EXTENSION cl_khr_fp64 : enable      \r\n"
    //--- OpenCL kernel function
    "__kernel void Test_GPU(__global double *data,      \r\n"
    "                        const   int N,              \r\n"
    "                        const   int total_arrays)   \r\n"
    "    {                                              \r\n"
    "        uint kernel_index=get_global_id(0);        \r\n"
    "        if (kernel_index>total_arrays) return;      \r\n"
    "        uint local_start_offset=kernel_index*N;    \r\n"
    "        for(int i=0; i<N; i++)                      \r\n"
    "        {                                          \r\n"
    "            data[i+local_start_offset] *= 2.0;      \r\n"
    "        }                                          \r\n"
    "    }                                              \r\n";

//+-----+
//| Test_CPU                                     |
//+-----+
bool Test_CPU(double &data[],const int N,const int id,const int total_arrays)
{
    //--- check array size
    if(ArraySize(data)==0) return(false);
    //--- check array index
    if(id>total_arrays) return(false);
    //--- calculate local offset for array with index id
    int local_start_offset=id*N;
    //--- multiply elements by 2
    for(int i=0; i<N; i++)
    {
        data[i+local_start_offset]*=2.0;
    }
    return true;
}

//---
#define ARRAY_SIZE   100 // size of the array
#define TOTAL_ARRAYS 5   // total arrays
//--- OpenCL handles
int cl_ctx; // OpenCL context handle
int cl_prg; // OpenCL program handle
int cl_krn; // OpenCL kernel handle
int cl_mem; // OpenCL buffer handle
//---
double dataArray1[]; // data array for CPU calculation
double dataArray2[]; // data array for GPU calculation
//+-----+
//| Script program start function                                     |
//+-----+

```



```

int OnStart()
{
//--- initialize OpenCL objects
//--- create OpenCL context
if((cl_ctx=CLContextCreate())==INVALID_HANDLE)
{
    Print("OpenCL not found. Error=",GetLastError());
    return(1);
}
//--- create OpenCL program
if((cl_prg=CLProgramCreate(cl_ctx,cl_src))==INVALID_HANDLE)
{
    CLContextFree(cl_ctx);
    Print("OpenCL program create failed. Error=",GetLastError());
    return(1);
}
//--- create OpenCL kernel
if((cl_krn=CLKernelCreate(cl_prg,"Test_GPU"))==INVALID_HANDLE)
{
    CLProgramFree(cl_prg);
    CLContextFree(cl_ctx);
    Print("OpenCL kernel create failed. Error=",GetLastError());
    return(1);
}
//--- create OpenCL buffer
if((cl_mem=CLBufferCreate(cl_ctx,ARRAY_SIZE*TOTAL_ARRAYS*sizeof(double),CL_MEM_READ_WRITE))!=INVALID_HANDLE)
{
    CLKernelFree(cl_krn);
    CLProgramFree(cl_prg);
    CLContextFree(cl_ctx);
    Print("OpenCL buffer create failed. Error=",GetLastError());
    return(1);
}
//--- set OpenCL kernel constant parameters
CLSetKernelArgMem(cl_krn,0,cl_mem);
CLSetKernelArg(cl_krn,1,ARRAY_SIZE);
CLSetKernelArg(cl_krn,2,TOTAL_ARRAYS);
//--- prepare data arrays
ArrayResize(DataArray1,ARRAY_SIZE*TOTAL_ARRAYS);
ArrayResize(DataArray2,ARRAY_SIZE*TOTAL_ARRAYS);
//--- fill arrays with data
for(int j=0; j<TOTAL_ARRAYS; j++)
{
    //--- calculate local start offset for jth array
    uint local_offset=j*ARRAY_SIZE;
    //--- prepare array with index j
    for(int i=0; i<ARRAY_SIZE; i++)
    {
        //--- fill arrays with function MathCos(i+j);
    }
}
}

```



```

        dataArray1[i+local_offset]=MathCos(i+j);
        dataArray2[i+local_offset]=MathCos(i+j);
    }
};

//--- test CPU calculation
for(int j=0; j<TOTAL_ARRAYS; j++)
{
    //--- calculation of the array with index j
    Test_CPU(dataArray1,ARRAY_SIZE,j,TOTAL_ARRAYS);
}

//--- prepare CLExecute params
uint offset[]={0};
//--- global work size
uint work[]={TOTAL_ARRAYS};
//--- write data to OpenCL buffer
CLBufferWrite(cl_mem,dataArray2);
//--- execute OpenCL kernel
CLExecute(cl_krn,1,offset,work);
//--- read data from OpenCL buffer
CLBufferRead(cl_mem,dataArray2);
//--- total error
double total_error=0;
//--- compare results and calculate error
for(int j=0; j<TOTAL_ARRAYS; j++)
{
    //--- calculate local offset for jth array
    uint local_offset=j*ARRAY_SIZE;
    //--- compare the results
    for(int i=0; i<ARRAY_SIZE; i++)
    {
        double v1=dataArray1[i+local_offset];
        double v2=dataArray2[i+local_offset];
        double delta=MathAbs(v2-v1);
        total_error+=delta;
        //--- show first and last arrays
        if((j==0) || (j==TOTAL_ARRAYS-1))
            PrintFormat("array %d of %d, element [%d]: %f, %f, [error]=%f",j+1,TOTAL_
    }
}

PrintFormat("Total error: %f",total_error);
//--- delete OpenCL objects
//--- free OpenCL buffer
CLBufferFree(cl_mem);
//--- free OpenCL kernel
CLKernelFree(cl_krn);
//--- free OpenCL program
CLProgramFree(cl_prg);
//--- free OpenCL context
CLContextFree(cl_ctx);

```



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


CLProgramFree

Removes an OpenCL program.

```
void CLProgramFree(  
    int program    // Handle to an OpenCL object  
);
```

Parameters

program

[in] Handle of the OpenCL object.

Return Value

None. In the case of an internal error the value of [_LastError](#) changes. For information about the error, use the [GetLastError\(\)](#) function.

CLKernelCreate

Creates the OpenCL program kernel and returns its handle.

```
int  CLKernelCreate(  
    int          program,          // Handle to an OpenCL object  
    const string kernel_name      // Kernel name  
);
```

Parameters

program

[in] Handle to an object of the OpenCL program.

kernel_name

[in] The name of the kernel function in the appropriate OpenCL program, in which execution begins.

Return Value

A handle to an OpenCL object if successful. In case of error -1 is returned. For information about the error, use the [GetLastError\(\)](#) function.

Note

At the moment, the following error codes are used:

- ERR_OPENCL_INVALID_HANDLE - invalid handle to OpenCL program.
- ERR_INVALID_PARAMETER - invalid string parameter.
- ERR_OPENCL_TOO_LONG_KERNEL_NAME - kernel name contains more than 127 characters.
- ERR_OPENCL_KERNEL_CREATE - internal error occurred while creating an OpenCL object.

CLKernelFree

Removes an OpenCL start function.

```
void CLKernelFree(  
    int kernel    // Handle to the kernel of an OpenCL program  
);
```

Parameters

kernel_name

[in] Handle of the kernel object.

Return Value

None. In the case of an internal error the value of [_LastError](#) changes. For information about the error, use the [GetLastError\(\)](#) function.

CLSetKernelArg

Sets a parameter for the OpenCL function.

```
bool CLSetKernelArg(  
    int    kernel,          // Handle to the kernel of an OpenCL program  
    uint   arg_index,       // The number of the argument of the OpenCL function  
    void   arg_value        // Source code  
);
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

arg_value

[in] The value of the function argument.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the [GetLastError\(\)](#) function.

Note

At the moment, the following error codes are used:

- ERR_INVALID_PARAMETER,
- ERR_OPENCL_INVALID_HANDLE - invalid handle to the OpenCL kernel.
- ERR_OPENCL_SET_KERNEL_PARAMETER - internal error of OpenCL.

CLSetKernelArgMem

Sets an OpenCL buffer as a parameter of the OpenCL function.

```
bool CLSetKernelArgMem(  
    int    kernel,           // Handle to the kernel of an OpenCL program  
    uint   arg_index,       // The number of the argument of the OpenCL function  
    int    cl_mem_handle     // Handle to OpenCL buffer  
);
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

cl_mem_handle

[in] A handle to an OpenCL buffer.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the [GetLastError\(\)](#) function.

CLSetKernelArgMemLocal

Sets the local buffer as an argument of the kernel function.

```
bool CLSetKernelArgMemLocal(  
    int    kernel,           // handle to a kernel of an OpenCL program  
    uint   arg_index,       // number of the OpenCL function argument  
    ulong  local_mem_size   // buffer size  
);
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

local_mem_size

[in] Buffer size in bytes.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the [GetLastError\(\)](#) function.

CLBufferCreate

Creates an OpenCL buffer and returns its handle.

```
int CLBufferCreate(  
    int    context,    // Handle to an OpenCL context  
    uint   size,       // Buffer size  
    uint   flags       // Flags combination which specify properties of OpenCL buffer  
);
```

Parameters

context

[in] A handle to context OpenCL.

size

[in] Buffer size in bytes.

flags

[in] Buffer properties that are set using a combination of flags: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY, CL_MEM_ALLOC_HOST_PTR.

Return Value

A handle to an OpenCL buffer if successful. In case of error -1 is returned. For information about the error, use the [GetLastError\(\)](#) function.

Note

At the moment, the following error codes are used:

- ERR_OPENCL_INVALID_HANDLE - invalid handle to OpenCL context.
- ERR_NOT_ENOUGH_MEMORY - insufficient memory.
- ERR_OPENCL_BUFFER_CREATE - internal error creating buffers.

CLBufferFree

Deletes an OpenCL buffer.

```
void CLBufferFree(  
    int    buffer    // Handle to an OpenCL buffer  
);
```

Parameters

buffer

[in] A handle to an OpenCL buffer.

Return Value

None. In the case of an internal error the value of [_LastError](#) changes. For information about the error, use the [GetLastError\(\)](#) function.

CLBufferWrite

Writes into the OpenCL buffer and returns the number of written elements.

```
uint CLBufferWrite(
    int          buffer,           // A handle to the OpenCL buffer
    const void&  data[],          // An array of values
    uint         buffer_offset=0,  // An offset in the OpenCL buffer in bytes,
    uint         data_offset=0,    // An offset in the array in elements, 0 by
    uint         data_count=WHOLE_ARRAY // The number of values from the array for
);
```

Parameters

buffer

[in] A handle of the OpenCL buffer.

data[]

[in] An array of values that should be written in the OpenCL buffer. Passed by reference.

buffer_offset

[in] An offset in the OpenCL buffer in bytes, from which writing begins. By default, writing start with the very beginning of the buffer.

data_offset

[in] The index of the first array element, starting from which values from the array are written in the OpenCL buffer. By default, values from the very beginning of the array are taken.

data_count

[in] The number of values that should be written. All the values of the array, by default.

Return Value

The number of written elements. 0 is returned in case of an error. For information about the error, use the [GetLastError\(\)](#) function.

Note

For one-dimensional arrays, the number of the element, with which reading of data for writing into an OpenCL buffer begins, is calculated taking into account the [AS_SERIES](#) flags.

An array of two or more dimensions is presented as one-dimensional. In this case, *data_offset* is the number of elements that should be skipped in the presentation, not the number of elements in the first dimension.

CLBufferRead

Reads an OpenCL buffer into an array and returns the number of read elements.

```
uint CLBufferRead(  
    int          buffer,           // A handle to the OpenCL buffer  
    const void&  data[],          // An array of values  
    uint         buffer_offset=0,  // An offset in the OpenCL buffer in bytes,  
    uint         data_offset=0,    // An offset in the array in elements, 0 by  
    uint         data_count=WHOLE_ARRAY // The number of values from the buffer for  
);
```

Parameters

buffer

[in] A handle of the OpenCL buffer.

data[]

[in] An array for receiving values from the OpenCL buffer. Passed by reference.

buffer_offset

[in] An offset in the OpenCL buffer in bytes, from which reading begins. By default, reading start with the very beginning of the buffer.

data_offset

[in] The index of the first array element for writing the values of the OpenCL buffer. By default, writing of the read values into an array starts from the zero index.

data_count

[in] The number of values that should be read. The whole OpenCL buffer is read by default.

Return Value

The number of read elements. 0 is returned in case of an error. For information about the error, use the [GetLastError\(\)](#) function.

Note

For one-dimensional arrays, the number of the element, into which writing of data into an OpenCL buffer begins, is calculated taking into account the [AS_SERIES](#) flags.

An array of two or more dimensions is presented as one-dimensional. In this case, *data_offset* is the number of elements that should be skipped in the presentation, not the number of elements in the first dimension.

CLExecute

The function runs an OpenCL program. There are 3 versions of the function:

1. Launching kernel functions using one kernel

```
bool CLExecute(
    int          kernel           // Handle to the kernel of an OpenCL program
);
```

2. Launching several kernel copies (OpenCL function) with task space description

```
bool CLExecute(
    int          kernel,           // Handle to the kernel of an OpenCL program
    uint         work_dim,         // Dimension of the tasks space
    const uint&  global_work_offset[], // Initial offset in the tasks space
    const uint&  global_work_size[]  // Total number of tasks
);
```

3. Launching several kernel copies (OpenCL function) with task space description and specification of the size of the group's local task subset

```
bool CLExecute(
    int          kernel,           // Handle to the kernel of an OpenCL program
    uint         work_dim,         // Dimension of the tasks space
    const uint&  global_work_offset[], // Initial offset in the tasks space
    const uint&  global_work_size[],  // Total number of tasks
    const uint&  local_work_size[]    // Number of tasks in the local group
);
```

Parameters

kernel

[in] Handle to the OpenCL kernel.

work_dim

[in] Dimension of the tasks space.

global_work_offset[]

[in] Initial offset in the tasks space.

global_work_size[]

[in] The size of a subset of tasks.

local_work_size[]

[in] The size of the group's local task subset.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the [GetLastError\(\)](#) function.

Note

Consider the use of the parameters in the following example:

- *work_dim* specifies the size of *work_items[]* array that describes the tasks. If *work_dim*=3, three-dimensional array *work_items[N1, N2, N3]* is used.
- *global_work_size[]* contains the values that set the *work_items[]* array size. If *work_dim*=3, *global_work_size[3]* array can be {40, 100, 320}. Then we have *work_items[40, 100, 320]*. So, the total number of tasks is $40 \times 100 \times 320 = 1\,280\,000$.
- *local_work_size[]* sets the subset of the tasks that will be executed by the specified kernel of OpenCL program. Its size is equal to *work_items[]* size and allows to split the common task subset into smaller subsets without loss of remainder in division. In fact, the size of *local_work_size[]* array should be selected so that the *work_items[]* global task set will be split into smaller subsets. In this example *local_work_size[3]={10, 10, 10}* will be OK, as *work_items[40, 100, 320]* can be gathered from *local_items[10, 10, 10]* array without division remainder.

CLExecutionStatus

Returns the OpenCL program execution status.

```
bool CLExecutionStatus(  
    int    kernel           // handle to a kernel of an OpenCL program  
);
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

Return Value

Returns the OpenCL program status. The value can be one of the following:

- CL_COMPLETE=0 - program complete,
- CL_RUNNING=1 - running,
- CL_SUBMITTED=2 - submitted for execution,
- CL_QUEUED=3 - queued,
- -1 (minus one) - error occurred when executing CLExecutionStatus().

Standard Library

This group of chapters contains the technical details of the MQL5 Standard Library and descriptions of all its key components.

MQL5 Standard Library is written in MQL5 and is designed to facilitate writing programs (indicators, scripts, experts) for end users. Library provides convenient access to the most of the internal MQL5 functions.

MQL5 Standard Library is placed in the working directory of the terminal in the 'Include' folder.

Section	Location
<u>Mathematics</u>	Include\Math\
<u>OpenCL</u>	Include\OpenCL\
<u>Basic Class CObject</u>	Include\
<u>Data Collections</u>	Include\Arrays\
<u>Files</u>	Include\Files\
<u>Strings</u>	Include\Strings\
<u>Graphic Objects</u>	Include\Objects\
<u>Custom Graphics</u>	Include\Canvas\
<u>Price Charts</u>	Include\Charts\
<u>Scientific Charts</u>	Include\Graphics\
<u>Indicators</u>	Include\Indicators\
<u>Trade classes</u>	Include\Trade\
<u>Strategy Modules</u>	Include\Expert\
<u>Panels and Dialogs</u>	Include\Controls\

Multiple libraries are provided to perform calculations in various areas of mathematics:

- [Statistics](#) - functions for working with various distributions of the Probability theory
- [Fuzzy logic](#) - library that implements Mamdani and Sugeno fuzzy inference systems
- [ALGLIB](#) - data analysis (clustering, decision trees, linear regression, neural networks), solving differential equations, Fourier transform, numerical integration, optimization problems, statistical analysis, and more.

Statistics

The Statistical Library provides a convenient way of working with the basic statistical distributions.

The library provides 5 functions for each distribution:

1. Calculation of probability density - functions of the form `MathProbabilityDensityX()`
2. Calculation of probabilities - functions of the form `MathCumulativeDistributionX()`
3. Calculation of distribution quantiles - functions of the form `MathQuantileX()`
4. Generation of random numbers with the specified distribution - functions of the form `MathRandomX()`
5. Calculation of the theoretical moments of the distributions - functions of the form `MathMomentsX()`

In addition to calculation of values for the individual random variables, the library also provides overloads for the functions, which perform the same calculations for arrays.

- [Statistical Characteristics](#)
- [Normal Distribution](#)
- [Log-normal distribution](#)
- [Beta distribution](#)
- [Noncentral beta distribution](#)
- [Gamma distribution](#)
- [Chi-squared distribution](#)
- [Noncentral chi-squared distribution](#)
- [Exponential distribution](#)
- [F-distribution](#)
- [Noncentral F-distribution](#)
- [t-distribution](#)
- [Noncentral t-distribution](#)
- [Logistic distribution](#)
- [Cauchy distribution](#)
- [Uniform distribution](#)
- [Weibull distribution](#)
- [Binomial distribution](#)
- [Negative binomial distribution](#)
- [Geometric distribution](#)
- [Hypergeometric distribution](#)
- [Poisson distribution](#)
- [Subfunctions](#)

Example:


```

//+-----+
//|                                     NormalDistributionExample.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- include the functions for calculating the normal distribution
#include <Math\Stat\Normal.mqh>
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- set the parameters of the normal distribution
double mu=5.0;
double sigma=1.0;
PrintFormat("Normal distribution with parameters mu=%G and sigma=%G, calculation ex
//--- set the interval
double x1=mu-sigma;
double x2=mu+sigma;
//--- variables for probability calculation
double cdf1,cdf2,probability;
//--- variables for error codes
int error_code1,error_code2;
//--- calculate the values of distribution functions
cdf1=MathCumulativeDistributionNormal(x1,mu,sigma,error_code1);
cdf2=MathCumulativeDistributionNormal(x2,mu,sigma,error_code2);
//--- check the error codes
if(error_code1==ERR_OK && error_code2==ERR_OK)
{
//--- calculate probability of a random variable in the range
probability=cdf2-cdf1;
//--- output the result
PrintFormat("1. Calculate probability of a random variable within the range of %
PrintFormat(" Answer: Probability = %5.8f",probability);
}

//--- Find the value range of random variable x, corresponding to the 95% confidence
probability=0.95; // set the confidence probability
//--- set the probabilities at the interval bounds
double p1=(1.0-probability)*0.5;
double p2=probability+(1.0-probability)*0.5;
//--- calculate the interval bounds
x1=MathQuantileNormal(p1,mu,sigma,error_code1);
x2=MathQuantileNormal(p2,mu,sigma,error_code2);
//--- check the error codes
if(error_code1==ERR_OK && error_code2==ERR_OK)
{
//--- output the result
PrintFormat("2. For confidence interval = %.2f, find the range of random variabl
PrintFormat(" Answer: range is %5.8f <= x <=%5.8f",x1,x2);
}

PrintFormat("3. Compute the first 4 calculated and theoretical moments of the dist
//--- Generate an array of random numbers, calculate the first 4 moments and compare v
int data_count=1000000; // set the number of values and prepare an array
double data[];
ArrayResize(data,data_count);
//--- generate random values and store them into the array

```



```

    for(int i=0; i<data_count; i++)
    {
        data[i]=MathRandomNormal(mu,sigma,error_code1);
    }
//--- set the index of the initial value and the amount of data for calculation
int start=0;
int count=data_count;
//--- calculate the first 4 moments of the generated values
double mean=MathMean(data,start,count);
double variance=MathVariance(data,start,count);
double skewness=MathSkewness(data,start,count);
double kurtosis=MathKurtosis(data,start,count);
//--- variables for the theoretical moments
double normal_mean=0;
double normal_variance=0;
double normal_skewness=0;
double normal_kurtosis=0;
//--- display the values of the calculated moments
PrintFormat("          Mean          Variance          Skewness          Kurtosis")
PrintFormat("Calculated  %.10f  %.10f  %.10f  %.10f",mean,variance,skewness,kurtosis);
//--- calculate the theoretical values of the moments and compare them with the obtained
if(MathMomentsNormal(mu,sigma,normal_mean,normal_variance,normal_skewness,normal_kurtosis))
{
    PrintFormat("Theoretical %.10f  %.10f  %.10f  %.10f",normal_mean,normal_variance,normal_skewness,normal_kurtosis);
    PrintFormat("Difference  %.10f  %.10f  %.10f  %.10f",mean-normal_mean,variance-normal_variance,skewness-normal_skewness,kurtosis-normal_kurtosis);
}
}

```


Statistical Characteristics

This group of functions calculates the Statistical Characteristics of the array elements:

- mean,
- variance,
- skewness,
- kurtosis,
- median,
- root-mean-square and
- standard deviation.

Function	Description
MathMean	Calculates the mean (first moment) of array elements
MathVariance	Calculates the variance (second moment) of array elements
MathSkewness	Calculates the skewness (third moment) of array elements
MathKurtosis	Calculates the kurtosis (fourth moment) of array elements
MathMoments	Calculates the first 4 moments (mean, variance, skewness, kurtosis) of array elements
MathMedian	Calculates the median value of array elements
MathStandardDeviation	Calculates the standard deviation of array elements
MathAverageDeviation	Calculates the average absolute deviation of array elements

MathMean

Calculates the mean (first moment) of array elements. Analog of the [mean\(\)](#) in R.

```
double MathMean(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation of the mean.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The mean of array elements. In case of error it returns [NaN](#) (not a number).

MathVariance

Calculates the variance (second moment) of array elements. Analog of the [var\(\)](#) in R.

```
double MathVariance(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Variance of array elements. In case of error it returns [NaN](#) (not a number).

MathSkewness

Calculates the skewness (third moment) of array elements. Analog of the [skewness\(\)](#) in R (e1071 library).

```
double MathSkewness(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Skewness of array elements. In case of error it returns [NaN](#) (not a number).

MathKurtosis

Calculates the kurtosis (fourth moment) of array elements. Analog of the [kurtosis\(\)](#) in R (e1071 library).

```
double MathKurtosis(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Kurtosis of array elements. In case of error it returns [NaN](#) (not a number).

Disclaimer

Calculation of the kurtosis is performed using the excess kurtosis around the normal distribution (excess kurtosis=kurtosis-3), i.e. the excess kurtosis of a normal distribution is zero.

It is positive if the peak of the distribution around the expected value is sharp, and negative if the peak is flat.

MathMoments

Calculates the first 4 moments (mean, variance, skewness, kurtosis) of array elements.

```
double MathMoments(
    const double& array[],           // array with data
    double& mean,                    // mean (1st moment)
    double& variance,                // variance (2nd moment)
    double& skewness,                // skewness (3rd moment)
    double& kurtosis,                // kurtosis (4th moment)
    const int start=0,               // initial index
    const int count=WHOLE_ARRAY     // the number of elements
);
```

Parameters

array

[in] Array with data for calculation.

mean

[out] Variable for the mean (1st moment)

variance

[out] Variable for the variance (2nd moment)

skewness

[out] Variable for the skewness (3rd moment)

kurtosis

[out] Variable for the kurtosis (4th moment)

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

Disclaimer

Calculation of the kurtosis is performed using the excess kurtosis around the normal distribution (excess kurtosis=kurtosis-3), i.e. the excess kurtosis of a normal distribution is zero.

It is positive if the peak of the distribution around the expected value is sharp, and negative if the peak is flat.

MathMedian

Calculates the median value of array elements. Analog of the [median\(\)](#) in R.

```
double MathMedian(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The median value of array elements. In case of error it returns [NaN](#) (not a number).

MathStandardDeviation

Calculates the standard deviation of array elements. Analog of the [sd\(\)](#) in R.

```
double MathStandardDeviation(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The standard deviation of array elements. In case of error it returns [NaN](#) (not a number).

MathAverageDeviation

Calculates the average absolute deviation of array elements. Analog of the [aad\(\)](#) in R.

```
double MathAverageDeviation(  
    const double& array[],           // array with data  
    const int     start=0,           // initial index  
    const int     count=WHOLE_ARRAY // the number of elements  
);
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The average absolute deviation of array elements. In case of error it returns [NaN](#) (not a number).

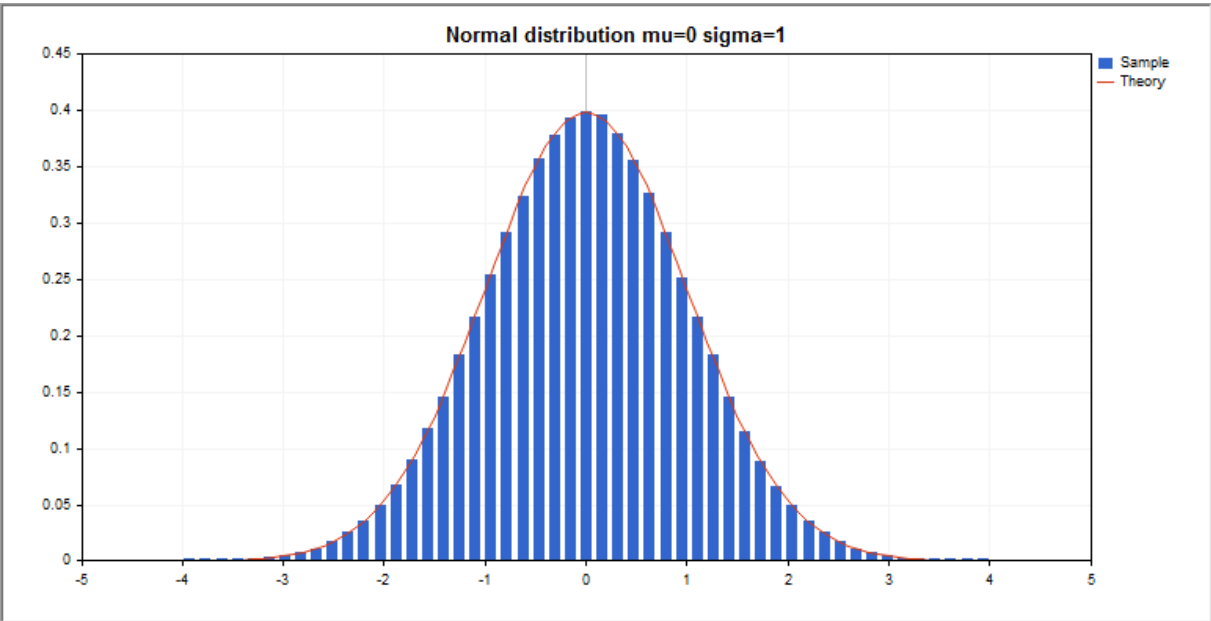
Normal Distribution

This section contains functions for working with normal distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the normal law. The distribution is defined by the following formula:

$$f_{Normal}(x|\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- x — value of the random variable
- μ — expected value
- σ — root-mean-square deviation



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNormal	Calculates the probability density function of the normal distribution
MathCumulativeDistributionNormal	Calculates the value of the normal probability distribution function
MathQuantileNormal	Calculates the value of the inverse normal distribution function for the specified probability
MathRandomNormal	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the normal law

MathMomentsNormal

Calculates the theoretical numerical values of the first 4 moments of the normal distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Normal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=0; // expected value (mean)
input double std_dev=1;    // root-mean-square deviation (standard deviation)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000; // the number of values in the sample
    int ncells=51; // the number of intervals in the histogram
    double x[]; // centers of the histogram intervals
    double y[]; // the number of values from the sample falling within the intervals
    double data[]; // sample of random values
    double max,min; // the maximum and minimum values in the sample
//--- obtain a sample from the normal distribution
    MathRandomNormal(mean_value,std_dev,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNormal(x2,mean_value,std_dev,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
        CGraphic graphic;
        if(ObjectFind(chart,name)<0)
            graphic.Create(chart,name,0,0,0,780,380);
        else
            graphic.Attach(chart,name);
        graphic.BackgroundMain(StringFormat("Normal distribution mu=%G sigma=%G",mean_value,
        graphic.BackgroundMainSize(16);
//--- plot all curves
        graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
        graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
//--- plot all curves
        graphic.CurvePlotAll();
        graphic.Update();
    }
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                            double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+

```



```
///| Calculates values for sequence generation |
///+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    ///--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    ///--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    ///--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityNormal

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNormal(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution (expected value)
    const double sigma,       // sigma parameter of the distribution (root-mean-square deviation)
    const bool log_mode,      // calculate the logarithm of the value
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNormal(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution (expected value)
    const double sigma,       // sigma parameter of the distribution (root-mean-square deviation)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dnorm\(\)](#) in R.

```
bool MathProbabilityDensityNormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // mean parameter of the distribution (expected value)
    const double sigma,       // sigma parameter of the distribution (root-mean-square deviation)
    const bool log_mode,      // calculate the logarithm of the value
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // mean parameter of the distribution (expected value)
    const double sigma,       // sigma parameter of the distribution (root-mean-square deviation)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of the probability density function.

MathCumulativeDistributionNormal

Calculates the value of the normal distribution function with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNormal(
    const double x,           // value of random variable
    const double mu,          // expected value
    const double sigma,       // root-mean-square deviation
    const bool tail,          // flag for calculation of tail
    const bool log_mode,      // calculate the logarithm of the value
    int& error_code           // variable to store the error code
);
```

Calculates the value of the normal distribution function with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNormal(
    const double x,           // value of random variable
    const double mu,          // expected value
    const double sigma,       // root-mean-square deviation
    int& error_code           // variable to store the error code
);
```

Calculates the value of the normal distribution function with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dnorm\(\)](#) in R.

```
bool MathCumulativeDistributionNormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // expected value
    const double sigma,       // root-mean-square deviation
    const bool tail,          // flag for calculation of tail
    const bool log_mode,      // calculate the logarithm of the value
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the normal distribution function with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionNormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // expected value
    const double sigma,       // root-mean-square deviation
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

tail

[in] Flag of calculation. If tail=true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of the probability function.

MathQuantileNormal

For the specified *probability*, the function calculates the value of inverse normal distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileNormal(
    const double probability, // probability value of random variable
    const double mu,         // expected value
    const double sigma,      // root-mean-square deviation
    const bool tail,         // flag for calculation of tail
    const bool log_mode,     // calculate the logarithm of the value
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse normal distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileNormal(
    const double probability, // probability value of random variable
    const double mu,         // expected value
    const double sigma,      // root-mean-square deviation
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse normal distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the [gnorm\(\)](#) in R.

```
bool MathQuantileNormal(
    const double& probability[], // array with probability values of random variable
    const double mu,            // expected value
    const double sigma,         // root-mean-square deviation
    const bool tail,            // flag for calculation of tail
    const bool log_mode,        // calculate the logarithm of the value
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse normal distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool MathQuantileNormal(
    const double& probability[], // array with probability values of random variable
    const double mu,            // expected value
    const double sigma,         // root-mean-square deviation
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

tail

[in] Flag of calculation. If false, then calculation is performed for 1.0 - probability.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the quantiles.

MathRandomNormal

Generates a pseudorandom variable distributed according to the normal law with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathRandomNormal (
    const double mu,           // expected value
    const double sigma,       // root-mean-square deviation
    int& error_code            // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the normal law with the mu and sigma parameters. In case of error it returns false. Analog of the [rnorm\(\)](#) in R.

```
bool MathRandomNormal (
    const double mu,           // expected value
    const double sigma,       // root-mean-square deviation
    const int data_count,     // amount of required data
    double& result[]          // array to obtain the pseudorandom variables
);
```

Parameters

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

data_count

[in] The number of pseudorandom variables to be obtained.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNormal

Calculates the theoretical numerical values of the first 4 moments of the normal distribution.

```
double MathMomentsNormal(  
    const double mu,           // expected value  
    const double sigma,       // root-mean-square deviation  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable to store the error code  
);
```

Parameters

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

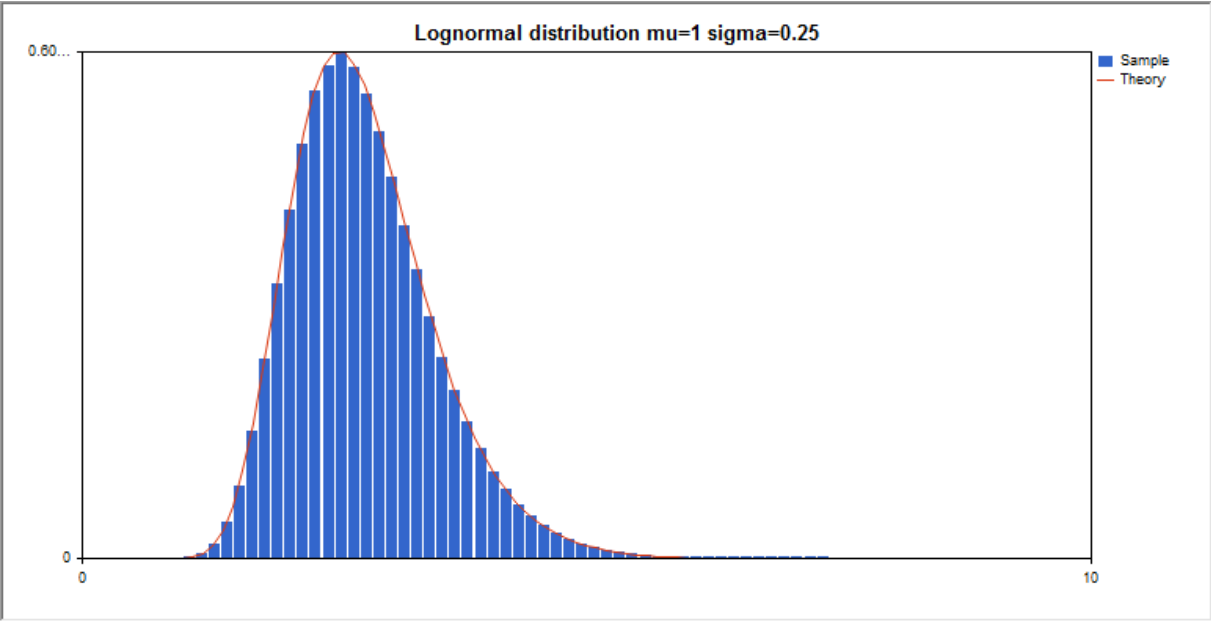
Log-normal distribution

This section contains functions for working with log-normal distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the log-normal law. The log-normal distribution is defined by the following formula:

$$f_{Lognormal}(x|\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$$

where:

- x – value of the random variable
- μ – logarithm of the expected value
- σ – logarithm of the root-mean-square deviation



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityLognormal	Calculates the probability density function of the log-normal distribution
MathCumulativeDistributionLognormal	Calculates the value of the log-normal probability distribution function
MathQuantileLognormal	Calculates the value of the inverse log-normal distribution function for the specified probability
MathRandomLognormal	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the log-normal law

MathMomentsLognormal

Calculates the theoretical numerical values of the first 4 moments of the log-normal distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Lognormal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=1.0; // logarithm of the expected value (log mean)
input double std_dev=0.25; // logarithm of the root-mean-square deviation (log standard deviation)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
MathSrand(GetTickCount());
//--- generate a sample of the random variable
long chart=0;
string name="GraphicNormal";
int n=1000000; // the number of values in the sample
int ncells=51; // the number of intervals in the histogram
double x[]; // centers of the histogram intervals
double y[]; // the number of values from the sample falling within the intervals
double data[]; // sample of random values
double max,min; // the maximum and minimum values in the sample
//--- obtain a sample from the log-normal distribution
MathRandomLognormal(mean_value,std_dev,n,data);
//--- calculate the data to plot the histogram
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityLognormal(x2,mean_value,std_dev,false,y2);
//--- set the scale
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
```



```

    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Lognormal distribution mu=%G sigma=%G",mean_va
graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(theor_max);
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency
    double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```



```
        frequency[ind]++;
    }
    return (true);
}

//+-----+
//|  Calculates values for sequence generation          |
//+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    //-- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //-- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    //-- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityLognormal

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityLognormal(
    const double x,           // value of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityLognormal(
    const double x,           // value of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns [NaN](#). Analog of the [dlnorm\(\)](#) in R.

```
bool MathProbabilityDensityLognormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log
    const bool log_mode,      // calculate the logarithm of the value, if log_mode
    double& result[]          // array for values of the probability density funct
);
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityLognormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log
    double& result[]          // array for values of the probability density funct
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of the probability density function.

MathCumulativeDistributionLognormal

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionLognormal(
    const double x,           // value of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=
    int& error_code           // variable to store the error code
);
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionLognormal(
    const double x,           // value of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    int& error_code           // variable to store the error code
);
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [plnorm\(\)](#) in R.

```
bool MathCumulativeDistributionLognormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if
    double& result[]          // array for values of the probability function
);
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionLognormal(
    const double& x[],        // array with the values of random variable
    const double mu,          // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log
    double& result[]          // array for values of the probability function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of the probability function.

MathQuantileLognormal

For the specified probability, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileLognormal(
    const double probability, // probability value of random variable occurrence
    const double mu,         // logarithm of the expected value (log mean)
    const double sigma,      // logarithm of the root-mean-square deviation (log s
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified probability, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileLognormal(
    const double probability, // probability value of random variable occurrence
    const double mu,         // logarithm of the expected value (log mean)
    const double sigma,      // logarithm of the root-mean-square deviation (log s
    int& error_code          // variable to store the error code
);
```

For the specified probability[] array of probability values, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the [qlnorm\(\)](#) in R.

```
bool MathQuantileLognormal(
    const double& probability[], // array with probability values of random variable
    const double mu,            // logarithm of the expected value (log mean)
    const double sigma,         // logarithm of the root-mean-square deviation (log
    const bool tail,            // flag of calculation, if false, then calculation :
    const bool log_mode,        // flag of calculation, if log_mode=true, calculatio
    double& result[]            // array with values of quantiles
);
```

For the specified probability[] array of probability values, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool MathQuantileLognormal(
    const double& probability[], // array with probability values of random variable
    const double mu,            // logarithm of the expected value (log mean)
    const double sigma,         // logarithm of the root-mean-square deviation (log
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable occurrence.

probability[]

[in] Array with probability values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to store the error code.

result[]

[out] Array with values of quantiles.

MathRandomLognormal

Generates a pseudorandom variable distributed according to the log-normal law with the mu sigma parameters. In case of error it returns [NaN](#).

```
double MathRandomLognormal (
    const double mu,           // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    int& error_code           // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the log-normal law with the mu sigma parameters. In case of error it returns false. Analog of the [rlnorm\(\)](#) in R.

```
double MathRandomLognormal (
    const double mu,           // logarithm of the expected value (log mean)
    const double sigma,       // logarithm of the root-mean-square deviation (log s
    const int data_count,     // amount of required data
    double& result[]          // array with values of pseudorandom variables
);
```

Parameters

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

data_count

[in] Amount of required data.

error_code

[out] Variable to store the error code.

result[]

[out] Array with values of pseudorandom variables.

MathMomentsLognormal

Calculates the theoretical numerical values of the first 4 moments of the log-normal distribution. Returns true if calculation of the moments has been successful, otherwise false.

```
double MathMomentsLognormal(  
    const double mu,           // logarithm of the expected value (log mean)  
    const double sigma,        // logarithm of the root-mean-square deviation (log s  
    double& mean,              // variable for the mean  
    double& variance,          // variable for the variance  
    double& skewness,          // variable for the skewness  
    double& kurtosis,          // variable for the kurtosis  
    int& error_code            // variable to store the error code  
);
```

Parameters

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

mean

[in] Variable for the mean.

variance

[out] Variable for the variance.

skewness

[out] Variable for the skewness.

kurtosis

[out] Variable for the kurtosis.

error code

[out] Variable to store the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

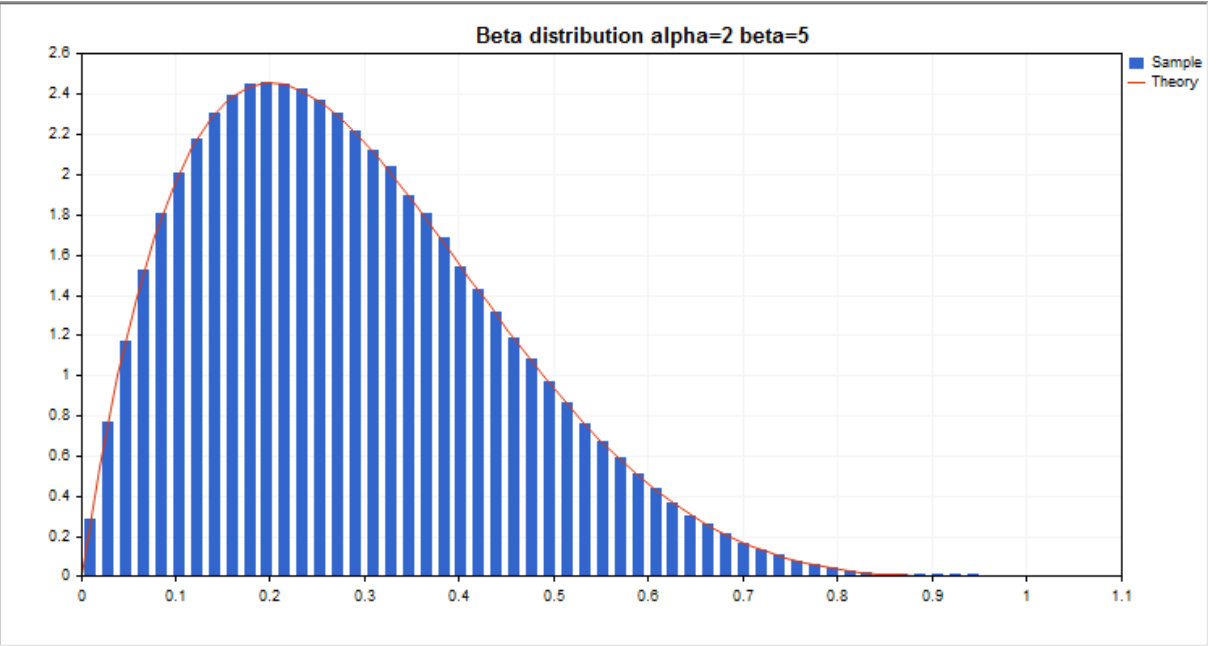
Beta distribution

This section contains functions for working with beta distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The beta distribution is defined by the following formula:

$$f_{Beta}(x|a,b) = \frac{1}{B(a,b)} x^{a-1} (1-x)^{b-1}$$

where:

- x – value of the random variable
- a – the first parameter of beta distribution
- b – the second parameter of beta distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityBeta	Calculates the probability density function of the beta distribution
MathCumulativeDistributionBeta	Calculates the value of the beta probability distribution function
MathQuantileBeta	Calculates the value of the inverse beta distribution function for the specified probability
MathRandomBeta	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the beta distribution law

MathMomentsBeta

Calculates the theoretical numerical values of the first 4 moments of the beta distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Beta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=2;    // the first parameter of beta distribution (shape1)
input double beta=5;     // the second parameter of beta distribution (shape2)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
    //--- initialize the random number generator
    MathSrand(GetTickCount());
    //--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the intervals
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
    //--- obtain a sample from the beta distribution
    MathRandomBeta(alpha,beta,n,data);
    //--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
    //--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityBeta(x2,alpha,beta,false,y2);
    //--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
        CGraphic graphic;
        if(ObjectFind(chart,name)<0)
            graphic.Create(chart,name,0,0,0,780,380);
        else
            graphic.Attach(chart,name);
        graphic.BackgroundMain(StringFormat("Beta distribution alpha=%G beta=%G",alpha,beta));
        graphic.BackgroundMainSize(16);
//--- plot all curves
        graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
        graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
        graphic.CurvePlotAll();
//--- plot all curves
        graphic.Update();
    }
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+

```



```
///| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
//--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
//--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityBeta

Calculates the value of the probability density function of beta distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dbeta\(\)](#) in R.

```
bool MathProbabilityDensityBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionBeta

Calculates the probability distribution function of beta distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of beta distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pbeta\(\)](#) in R.

```
bool MathCumulativeDistributionBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileBeta

For the specified *probability*, the function calculates the value of inverse beta distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileBeta(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of beta distribution (shape1)
    const double b,          // the second parameter of beta distribution (shape2)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse beta distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileBeta(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of beta distribution (shape1)
    const double b,          // the second parameter of beta distribution (shape2)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse beta distribution function with the a and b parameters. In case of error it returns false. Analog of the [qbeta\(\)](#) in R.

```
double MathQuantileBeta(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of beta distribution (shape1)
    const double b,             // the second parameter of beta distribution (shape2)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse beta distribution function with the a and b parameters. In case of error it returns false.

```
bool MathQuantileBeta(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of beta distribution (shape1)
    const double b,             // the second parameter of beta distribution (shape2)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

tail

[in] Flag of calculation, if lower_tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomBeta

Generates a pseudorandom variable distributed according to the law of beta distribution with the *a* and *b* parameters. In case of error it returns [NaN](#).

```
double MathRandomBeta(  
    const double a,           // the first parameter of beta distribution (shape1)  
    const double b,           // the second parameter of beta distribution (shape2)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of beta distribution with the *a* and *b* parameters. In case of error it returns false. Analog of the [rbeta\(\)](#) in R.

```
bool MathRandomBeta(  
    const double a,           // the first parameter of beta distribution (shape1)  
    const double b,           // the second parameter of beta distribution (shape2)  
    const int data_count,     // amount of required data  
    double& result[]          // array to obtain the pseudorandom variables  
);
```

Parameters

a

[in] The first parameter of beta distribution (shape1)

b

[in] The second parameter of beta distribution (shape2).

data_count

[in] The number of pseudorandom variables to be obtained.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsBeta

Calculates the theoretical numerical values of the first 4 moments of the beta distribution.

```
double MathMomentsBeta(  
    const double a,           // the first parameter of beta distribution (shape1)  
    const double b,           // the second parameter of beta distribution (shape2)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

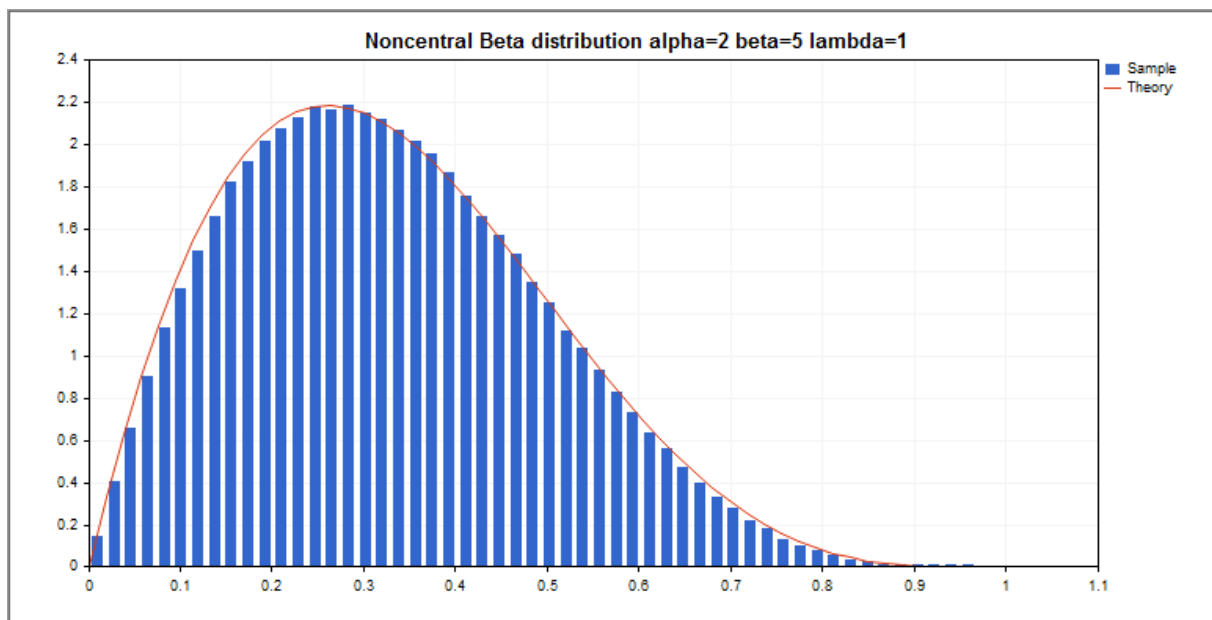
Noncentral beta distribution

This section contains functions for working with noncentral beta distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The noncentral beta distribution is defined by the following formula:

$$f_{\text{NoncentralBeta}}(x|a,b,\lambda) = \sum_{r=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^r}{r!} \frac{x^{a+r-1}(1-x)^{b-1}}{B(a+r,b)}$$

where:

- x — value of the random variable
- a — the first parameter of beta distribution
- b — the second parameter of beta distribution
- λ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
<u>MathProbabilityDensityNoncentralBeta</u>	Calculates the probability density function of the noncentral beta distribution
<u>MathCumulativeDistributionNoncentralBeta</u>	Calculates the value of the noncentral beta probability distribution function
<u>MathQuantileNoncentralBeta</u>	Calculates the value of the inverse noncentral beta distribution function for the specified probability
<u>MathRandomNoncentralBeta</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according

	to the noncentral beta distribution law
<u>MathMomentsNoncentralBeta</u>	Calculates the theoretical numerical values of the first 4 moments of the noncentral beta distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralBeta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=2;    // the first parameter of beta distribution (shape1)
input double b_par=5;    // the second parameter of beta distribution (shape2)
input double l_par=1;    // noncentrality parameter (lambda)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=53;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral beta distribution
    MathRandomNoncentralBeta(a_par,b_par,l_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralBeta(x2,a_par,b_par,l_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];

```



```

double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral Beta distribution alpha=%G beta=%G
                                     a_par,b_par,l_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```



```
    }
    return (true);
}
//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityNoncentralBeta

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dbeta\(\)](#) in R.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

lambda

[in] Noncentrality parameter

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralBeta

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pbeta\(\)](#) in R.

```
bool MathCumulativeDistributionNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    double& result[]          // array for values of the probability function
);
```


Parameters*x*

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

lambda

[in] Noncentrality parameter

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralBeta

Calculates the value of the inverse probability distribution function of noncentral beta distribution with the *a*, *b* and *lambda* parameters for the occurrence probability of a random variable *probability*. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralBeta(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of beta distribution (shape1)
    const double b,          // the second parameter of beta distribution (shape2)
    const double lambda,     // noncentrality parameter
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

Calculates the value of the inverse probability distribution function of noncentral beta distribution with the *a*, *b* and *lambda* parameters for the occurrence probability of a random variable *probability*. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralBeta(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of beta distribution (shape1)
    const double b,          // the second parameter of beta distribution (shape2)
    const double lambda,     // noncentrality parameter
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of the inverse probability distribution function of noncentral beta distribution with the *a*, *b* and *lambda* parameters. In case of error it returns false. Analog of the [qbeta\(\)](#) in R.

```
double MathQuantileNoncentralBeta(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of beta distribution (shape1)
    const double b,             // the second parameter of beta distribution (shape2)
    const double lambda,        // noncentrality parameter
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of the inverse probability distribution function of noncentral beta distribution with the *a*, *b* and *lambda* parameters. In case of error it returns false.

```
bool MathQuantileNoncentralBeta(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of beta distribution (shape1)
    const double b,             // the second parameter of beta distribution (shape2)
```



```
const double  lambda,           // noncentrality parameter
double&       result[]         // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralBeta

Generates a pseudorandom variable distributed according to the law of noncentral beta distribution the *a*, *b* and *lambda* parameters. In case of error it returns [NaN](#).

```
double MathRandomNoncentralBeta (
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the law of noncentral beta distribution the *a*, *b* and *lambda* parameters. In case of error it returns false. Analog of the [rbeta\(\)](#) in R.

```
bool MathRandomNoncentralBeta (
    const double a,           // the first parameter of beta distribution (shape1)
    const double b,           // the second parameter of beta distribution (shape2)
    const double lambda,      // noncentrality parameter
    const int data_count,     // amount of required data
    double& result[]         // array to obtain the pseudorandom variables
);
```

Parameters

a

[in] The first parameter of beta distribution (shape1)

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralBeta

Calculates the theoretical numerical values of the first 4 moments of the noncentral beta distribution with the *a*, *b* and *lambda* parameters.

```
double MathMomentsNoncentralBeta(  
    const double a,           // the first parameter of beta distribution (shape1)  
    const double b,           // the second parameter of beta distribution (shape2)  
    const double lambda,      // noncentrality parameter  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

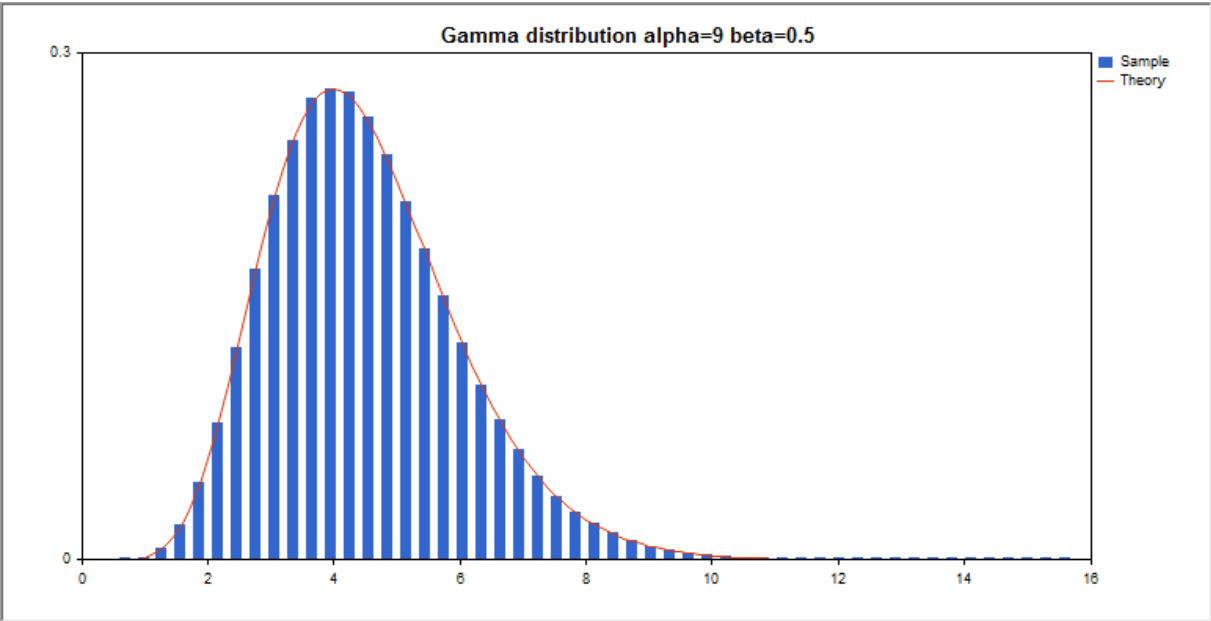
Gamma distribution

This section contains functions for working with gamma distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The gamma distribution is defined by the following formula:

$$f_{Gamma}(x|a,b) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{-\frac{x}{b}}$$

where:

- x – value of the random variable
- a – the first parameter of distribution
- b – the second parameter of distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityGamma	Calculates the probability density function of the gamma distribution
MathCumulativeDistributionGamma	Calculates the value of the gamma probability distribution function
MathQuantileGamma	Calculates the value of the inverse gamma distribution function for the specified probability
MathRandomGamma	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the gamma distribution law

MathMomentsGamma

Calculates the theoretical numerical values of the first 4 moments of the gamma distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Gamma.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=9;    // the first parameter of distribution (shape)
input double beta=0.5;   // the second parameter of distribution (scale)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
    //--- initialize the random number generator
    MathSrand(GetTickCount());
    //--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the intervals
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
    //--- obtain a sample from the gamma distribution
    MathRandomGamma(alpha,beta,n,data);
    //--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
    //--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityGamma(x2,alpha,beta,false,y2);
    //--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);
    else
        graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("Gamma distribution alpha=%G beta=%G",alpha,beta));
    graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
    graphic.YAxis().AutoScale(false);
    graphic.YAxis().Max(NormalizeDouble(theor_max,1));
    graphic.YAxis().Min(0);
//--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
    graphic.CurvePlotAll();
//--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```



```
    }
    return (true);
}
//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityNoncentralBeta

Calculates the value of the probability density function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // value of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dgamma\(\)](#) in R.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionGamma

Calculates the probability distribution function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionGamma(
    const double x,           // value of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionGamma(
    const double x,           // value of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pgamma\(\)](#) in R.

```
bool MathCumulativeDistributionGamma(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability function
);
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionGamma(
    const double& x[],        // array with the values of random variable
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    double& result[]         // array for values of the probability function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale)

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileGamma

For the specified *probability*, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileGamma(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of the distribution (shape)
    const double b,          // the second parameter of the distribution (scale)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileGamma(
    const double probability, // probability value of random variable occurrence
    const double a,          // the first parameter of the distribution (shape)
    const double b,          // the second parameter of the distribution (scale)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns false. Analog of the [qgamma\(\)](#) in R.

```
double MathQuantileGamma(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of the distribution (shape)
    const double b,             // the second parameter of the distribution (scale)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns false.

```
bool MathQuantileGamma(
    const double& probability[], // array with probability values of random variable
    const double a,             // the first parameter of the distribution (shape)
    const double b,             // the second parameter of the distribution (scale)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomGamma

Generates a pseudorandom variable distributed according to the law of gamma distribution with the *a* and *b* parameters. In case of error it returns [NaN](#).

```
double MathRandomGamma (
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    int& error_code           // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the law of gamma distribution with the *a* and *b* parameters. In case of error it returns false. Analog of the [rgamma\(\)](#) in R.

```
bool MathRandomGamma (
    const double a,           // the first parameter of the distribution (shape)
    const double b,           // the second parameter of the distribution (scale)
    const int data_count,     // amount of required data
    double& result[]          // array with values of pseudorandom variables
);
```

Parameters

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsGamma

Calculates the theoretical numerical values of the first 4 moments of the gamma distribution with the a and b parameters.

```
double MathMomentsGamma(  
    const double a,           // the first parameter of the distribution (shape)  
    const double b,           // the second parameter of the distribution (scale)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

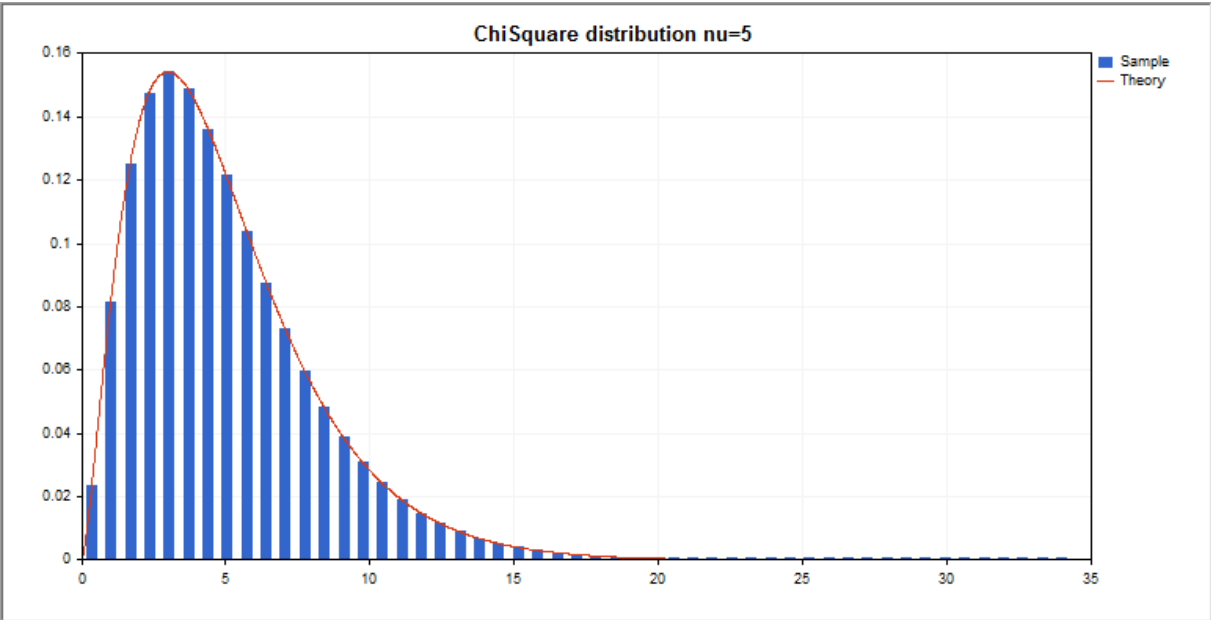
Chi-squared distribution

This section contains functions for working with chi-squared distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The chi-squared distribution is defined by the following formula:

$$f_{Chi-Square}(x|\nu) = \frac{x^{\frac{(\nu-2)}{2}} e^{-\frac{x}{2}}}{2^{\frac{\nu}{2}} \Gamma\left(\frac{\nu}{2}\right)}$$

where:

- x – value of the random variable
- ν – number of degrees of freedom



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityChiSquare	Calculates the probability density function of the chi-squared distribution
MathCumulativeDistributionChiSquare	Calculates the value of the chi-squared probability distribution function
MathQuantileChiSquare	Calculates the value of the inverse chi-squared distribution function for the specified probability
MathRandomChiSquare	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the chi-squared distribution law

MathMomentsChiSquare

Calculates the theoretical numerical values of the first 4 moments of the chi-squared distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\ChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=5;    // the number of degrees of freedom
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the chi-squared distribution
    MathRandomChiSquare(nu_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityChiSquare(x2,nu_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);
    else
        graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("ChiSquare distribution nu=%G ",nu_par));
    graphic.BackgroundMainSize(16);
//--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
    graphic.CurvePlotAll();
//--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+

```



```
///| Calculates values for sequence generation |
///+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    ///--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    ///--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    ///--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityChiSquare

Calculates the value of the probability density function of chi-squared distribution with the *nu* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of chi-squared distribution with the *nu* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of chi-squared distribution with the *nu* parameter for an array of random variables *x[]*. In case of error it returns false. Analog of the [dchisq\(\)](#) in R.

```
bool MathProbabilityDensityChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability density function of chi-squared distribution with the *nu* parameter for an array of random variables *x[]*. In case of error it returns false.

```
bool MathProbabilityDensityChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom)

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionChiSquare

Calculates the probability distribution function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionChiSquare (
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns [NaN](#)

```
double MathCumulativeDistributionChiSquare (
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the [pchisq\(\)](#) in R.

```
bool MathCumulativeDistributionChiSquare (
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability function
);
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionChiSquare (
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    double& result[]         // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileChiSquare

For the specified *probability*, the function calculates the value of inverse chi-squared distribution function. In case of error it returns [NaN](#).

```
double MathQuantileChiSquare(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse chi-squared distribution function. In case of error it returns [NaN](#).

```
double MathQuantileChiSquare(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse chi-squared distribution function. In case of error it returns false. Analog of the [gchisq\(\)](#) in R.

```
double MathQuantileChiSquare(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse chi-squared distribution function. In case of error it returns false.

```
bool MathQuantileChiSquare(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomChiSquare

Generates a pseudorandom variable distributed according to the law of chi-squared distribution with the *nu* parameter. In case of error it returns [NaN](#).

```
double MathRandomChiSquare (
    const double  nu,           // parameter of distribution (number of degrees of freedom)
    int&          error_code     // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the law of chi-squared distribution with the *nu* parameter. In case of error it returns false. Analog of the [rchisq\(\)](#) in R.

```
bool MathRandomChiSquare (
    const double  nu,           // parameter of distribution (number of degrees of freedom)
    const int     data_count,    // amount of required data
    double&       result[]      // array with values of pseudorandom variables
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsChiSquare

Calculates the theoretical numerical values of the first 4 moments of the chi-squared distribution with the nu parameter.

```
double MathMomentsChiSquare(  
    const double  nu,           // parameter of distribution (number of degrees of fr  
    double&       mean,         // variable for the mean  
    double&       variance,     // variable for the variance  
    double&       skewness,     // variable for the skewness  
    double&       kurtosis,     // variable for the kurtosis  
    int&         error_code     // variable for the error code  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

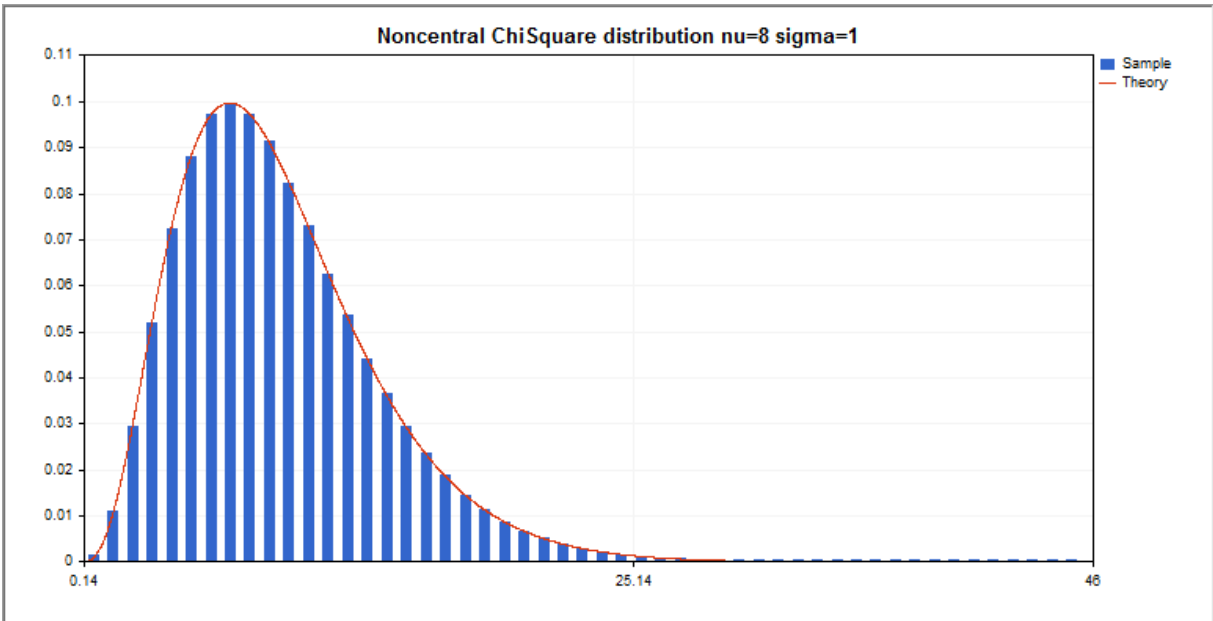
Noncentral chi-squared distribution

This section contains functions for working with noncentral chi-squared distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The noncentral chi-squared distribution is defined by the following formula:

$$f_{NoncentralChiSquare}(x|v,\sigma) = \frac{1}{2^{\frac{v}{2}}\Gamma(\frac{1}{2})} x^{\frac{v}{2}-1} e^{-\frac{(x+\sigma)}{2}} \sum_{r=0}^{\infty} \frac{(\lambda x)^r}{(2r)!} \frac{\Gamma(\frac{1}{2}+r)}{\Gamma(\frac{v}{2}+r)}$$

where:

- x – value of the random variable
- v – number of degrees of freedom
- σ – noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralChiSquare	Calculates the probability density function of the noncentral chi-squared distribution
MathCumulativeDistributionNoncentralChiSquare	Calculates the value of the noncentral chi-squared probability distribution function
MathQuantileNoncentralChiSquare	Calculates the value of the inverse noncentral chi-squared distribution function for the specified probability
MathRandomNoncentralChiSquare	Generates a pseudorandom variable/array of pseudorandom variables distributed according

	to the noncentral chi-squared distribution law
<u>MathMomentsNoncentralChiSquare</u>	Calculates the theoretical numerical values of the first 4 moments of the noncentral chi-squared distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=8;    // the number of degrees of freedom
input double si_par=1;    // noncentrality parameter
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral chi-squared distribution
    MathRandomNoncentralChiSquare(nu_par,si_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralChiSquare(x2,nu_par,si_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];

```



```

double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral ChiSquare distribution nu=%G sigma=
graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
graphic.XAxis().AutoScale(false);
graphic.XAxis().Max(NormalizeDouble(max,0));
graphic.XAxis().Min(min);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);

```



```

        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

//+-----+
//|  Calculates values for sequence generation          |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}

```


MathProbabilityDensityNoncentralChiSquare

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dchisq\(\)](#) in R.

```
bool MathProbabilityDensityNoncentralChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNoncentralChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    double& result[]         // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralChiSquare

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool tail,          // flag of calculation, if lower_tail=true, then the probability is calculated for the lower tail
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralChiSquare(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pchisq\(\)](#) in R.

```
bool MathCumulativeDistributionNoncentralChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool tail,          // flag of calculation, if lower_tail=true, then the probability is calculated for the lower tail
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionNoncentralChiSquare(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    double& result[]          // array for values of the probability function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralChiSquare

For the specified *probability*, the function calculates the value of inverse noncentral chi-squared distribution function with the *nu* and *sigma* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralChiSquare(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const double sigma,      // noncentrality parameter
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse noncentral chi-squared distribution function with the *nu* and *sigma* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralChiSquare(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const double sigma,      // noncentrality parameter
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral chi-squared distribution function with the *nu* and *sigma* parameters. In case of error it returns false. Analog of the [qchisq\(\)](#) in R.

```
double MathQuantileNoncentralChiSquare(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const double sigma,         // noncentrality parameter
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral chi-squared distribution function. In case of error it returns false.

```
bool MathQuantileNoncentralChiSquare(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const double sigma,         // noncentrality parameter
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralChiSquare

Generates a pseudorandom variable distributed according to the law of noncentral chi-squared distribution with the *nu* and *sigma* parameters. In case of error it returns [NaN](#).

```
double MathRandomNoncentralChiSquare(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double sigma,        // noncentrality parameter  
    int& error_code            // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of noncentral chi-squared distribution with the *nu* and *sigma* parameters. In case of error it returns false. Analog of the [rchisq\(\)](#) in R.

```
bool MathRandomNoncentralChiSquare(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double sigma,        // noncentrality parameter  
    const int data_count,      // amount of required data  
    double& result[]           // array with values of pseudorandom variables  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralChiSquare

Calculates the theoretical numerical values of the first 4 moments of the noncentral chi-squared distribution with the nu and sigma parameters.

```
double MathMomentsNoncentralChiSquare(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double sigma,        // noncentrality parameter  
    double& mean,              // variable for the mean  
    double& variance,          // variable for the variance  
    double& skewness,          // variable for the skewness  
    double& kurtosis,          // variable for the kurtosis  
    int& error_code             // variable for the error code  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

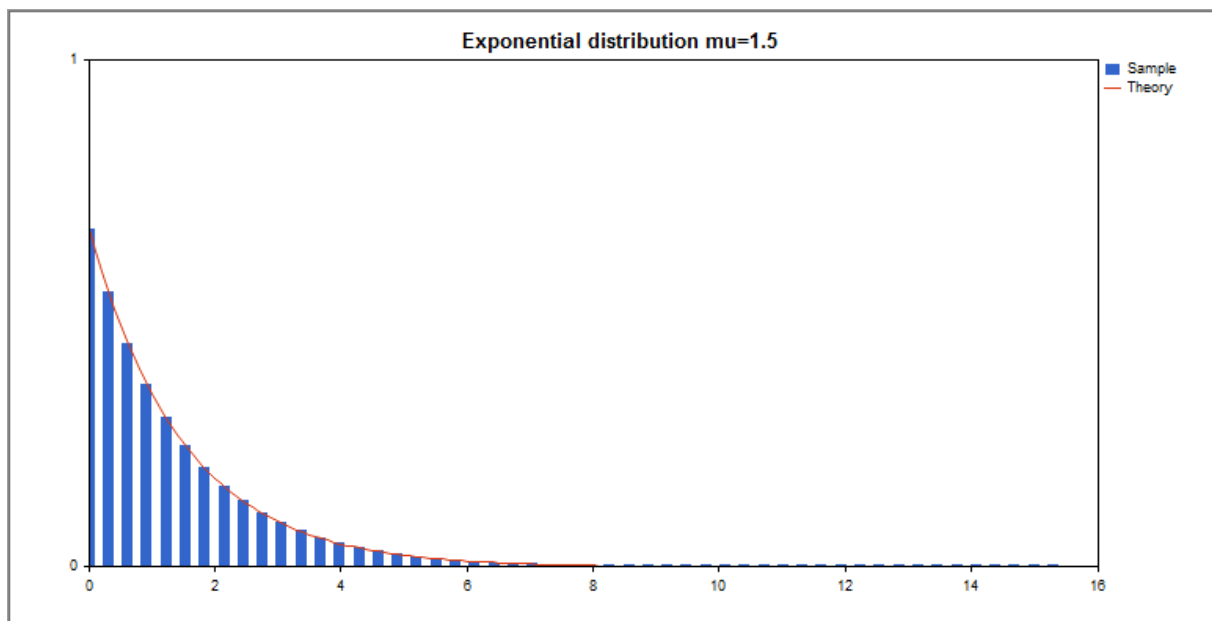
Exponential

This section contains functions for working with exponential distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of exponential distribution. The exponential distribution is defined by the following formula:

$$f_{\text{Exponential}}(x|\mu) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$$

where:

- x — value of the random variable
- μ — expected value



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
<u>MathProbabilityDensityExponential</u>	Calculates the probability density function of the exponential distribution
<u>MathCumulativeDistributionExponential</u>	Calculates the value of the exponential probability distribution function
<u>MathQuantileExponential</u>	Calculates the value of the inverse exponential distribution function for the specified probability
<u>MathRandomExponential</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the law of exponential distribution
<u>MathMomentsExponential</u>	Calculates the theoretical numerical values of the first 4 moments of the exponential

distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\Exponential.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=1.5;    // the number of degrees of freedom
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the exponential distribution
    MathRandomExponential(mu_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityExponential(x2,mu_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts

```



```

CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Exponential distribution mu=%G ",mu_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |
//+-----+

```



```
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityExponential

Calculates the value of the probability density function of exponential distribution with the μ parameter for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityExponential(
    const double x,           // value of random variable
    const double mu,          // parameter of the distribution (expected value)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of exponential distribution with the μ parameter for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityExponential(
    const double x,           // value of random variable
    const double mu,          // parameter of the distribution (expected value)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of exponential distribution with the μ parameter for an array of random variables $x[]$. In case of error it returns false. Analog of the [dexp\(\)](#) in R.

```
bool MathProbabilityDensityExponential(
    const double& x[],         // array with the values of random variable
    const double mu,           // parameter of the distribution (expected value)
    const bool log_mode,       // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]           // array for values of the probability density function
);
```

Calculates the value of the probability density function of exponential distribution with the μ parameter for an array of random variables $x[]$. In case of error it returns false.

```
bool MathProbabilityDensityExponential(
    const double& x[],         // array with the values of random variable
    const double mu,           // parameter of the distribution (expected value)
    double& result[]           // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

$x[]$

[in] Array with the values of random variable.

μ

[in] Parameter of the distribution (expected value)

\log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionExponential

Calculates the exponential distribution function of probabilities with the mu parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionExponential(
    const double x,           // value of random variable
    const double mu,          // parameter of the distribution (expected value)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the exponential distribution function of probabilities with the mu parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionExponential(
    const double x,           // value of random variable
    const double mu,          // parameter of the distribution (expected value)
    int& error_code           // variable to store the error code
);
```

Calculates the exponential distribution function of probabilities with the mu parameter for an array of random variables x[]. In case of error it returns false. Analog of the [pexp\(\)](#) in R.

```
bool MathCumulativeDistributionExponential(
    const double& x[],        // array with the values of random variable
    const double mu,          // parameter of the distribution (expected value)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability function
);
```

Calculates the exponential distribution function of probabilities with the mu parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionExponential(
    const double& x[],        // array with the values of random variable
    const double mu,          // parameter of the distribution (expected value)
    double& result[]         // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Parameter of the distribution (expected value).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileExponential

For the specified *probability*, the function calculates the value of inverse exponential distribution function with the *mu* parameter. In case of error it returns [NaN](#).

```
double MathQuantileExponential(
    const double probability, // probability value of random variable occurrence
    const double mu,         // parameter of the distribution (expected value)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse exponential distribution function with the *mu* parameter. In case of error it returns [NaN](#).

```
double MathQuantileExponential(
    const double probability, // probability value of random variable occurrence
    const double mu,         // parameter of the distribution (expected value)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse exponential distribution function with the *mu* parameter. In case of error it returns false. Analog of the [qexp\(\)](#) in R.

```
double MathQuantileExponential(
    const double& probability[], // array with probability values of random variable
    const double mu,            // parameter of the distribution (expected value)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse exponential distribution function with the *mu* parameter. In case of error it returns false.

```
bool MathQuantileExponential(
    const double& probability[], // array with probability values of random variable
    const double mu,            // parameter of the distribution (expected value)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] Parameter of the distribution (expected value).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomExponential

Generates a pseudorandom variable distributed according to the law of exponential distribution with the μ parameter. In case of error it returns [NaN](#).

```
double MathRandomExponential(  
    const double mu,           // parameter of the distribution (expected value)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of exponential distribution with the μ parameter. In case of error it returns false. Analog of the [rexp\(\)](#) in R.

```
bool MathRandomExponential(  
    const double mu,           // parameter of the distribution (expected value)  
    const int data_count,      // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

mu

[in] Parameter of the distribution (expected value).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsExponential

Calculates the theoretical numerical values of the first 4 moments of the exponential distribution with the mu parameter.

```
double MathMomentsExponential(  
    const double mu,           // parameter of the distribution (expected value)  
    double& mean,              // variable for the mean  
    double& variance,          // variable for the variance  
    double& skewness,          // variable for the skewness  
    double& kurtosis,          // variable for the kurtosis  
    int& error_code            // variable for the error code  
);
```

Parameters

mu

[in] Parameter of the distribution (expected value).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

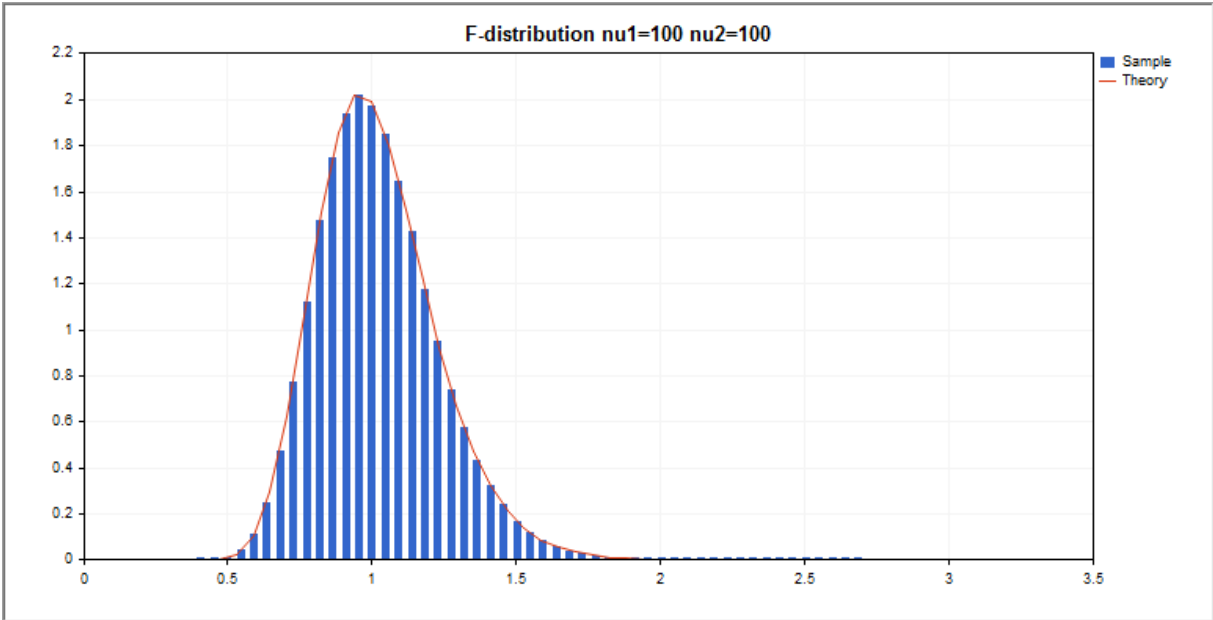
F-distribution

This section contains functions for working with F-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of Fisher's F-distribution. The F-distribution is defined by the following formula:

$$f_F(x|v_1,v_2)=\frac{\Gamma\left(\frac{v_1+v_2}{2}\right)}{\Gamma\left(\frac{v_1}{2}\right)\Gamma\left(\frac{v_2}{2}\right)}\left(\frac{v_1}{v_2}\right)^{\frac{v_1}{2}}\frac{x^{\frac{v_1-2}{2}}}{\left(1+\left(\frac{v_1}{v_2}\right)x\right)^{\frac{v_1+v_2}{2}}}$$

where:

- x — value of the random variable
- v1 — the first parameter of distribution (number of degrees of freedom)
- v2 — the second parameter of distribution (number of degrees of freedom)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityF	Calculates the probability density function of the F-distribution
MathCumulativeDistributionF	Calculates the value of the F-distribution function
MathQuantileF	Calculates the value of the inverse F-distribution function for the specified probability

<u>MathRandomF</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Fisher's law
<u>MathMomentsF</u>	Calculates the theoretical numerical values of the first 4 moments of the Fisher's F-distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\F.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=100;    // the first number of degrees of freedom
input double nu_2=100;    // the second number of degrees of freedom
//+-----+
//| Script program start function |
//+-----+
void OnStart ()
{
    //--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
    //--- initialize the random number generator
    MathSrand(GetTickCount());
    //--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=51;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the int
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
    //--- obtain a sample from the Fisher's F-distribution
    MathRandomF(nu_1,nu_2,n,data);
    //--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
    //--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityF(x2,nu_1,nu_2,false,y2);
    //--- set the scale

```



```

double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("F-distribution nu1=%G nu2=%G",nu_1,nu_2));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(4);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```



```
    }
    return (true);
}
//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityF

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false. Analog of the [df\(\)](#) in R.

```
bool MathProbabilityDensityF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionF

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding the value is calculated
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pf\(\)](#) in R.

```
bool MathCumulativeDistributionF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding the value is calculated
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability function
);
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    double& result[]         // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileF

For the specified *probability*, the function calculates the value of inverse Fisher's F-distribution function with the *nu1* and *nu2* parameters. In case of error it returns [NaN](#).

```
double MathQuantileF(
    const double probability, // probability value of random variable occurrence
    const double nu1,        // the first parameter of distribution (number of degrees of freedom)
    const double nu2,        // the second parameter of distribution (number of degrees of freedom)
    const bool tail,         // flag of calculation, if false, then calculation is for the left tail
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation is for the logarithm of the function
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse Fisher's F-distribution function with the *nu1* and *nu2* parameters. In case of error it returns [NaN](#).

```
double MathQuantileF(
    const double probability, // probability value of random variable occurrence
    const double nu1,        // the first parameter of distribution (number of degrees of freedom)
    const double nu2,        // the second parameter of distribution (number of degrees of freedom)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Fisher's F-distribution function with the *nu1* and *nu2* parameters. In case of error it returns false. Analog of the [qf\(\)](#) in R.

```
double MathQuantileF(
    const double& probability[], // array with probability values of random variable
    const double nu1,           // the first parameter of distribution (number of degrees of freedom)
    const double nu2,           // the second parameter of distribution (number of degrees of freedom)
    const bool tail,            // flag of calculation, if false, then calculation is for the left tail
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation is for the logarithm of the function
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Fisher's F-distribution function with the *nu1* and *nu2* parameters. In case of error it returns false.

```
bool MathQuantileF(
    const double& probability[], // array with probability values of random variable
    const double nu1,           // the first parameter of distribution (number of degrees of freedom)
    const double nu2,           // the second parameter of distribution (number of degrees of freedom)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation, if lower_tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomF

Generates a pseudorandom variable distributed according to the law of Fisher's F-distribution with the *nu1* and *nu2* parameters. In case of error it returns [NaN](#).

```
double MathRandomF(  
    const double nu1,           // the first parameter of distribution (number of de  
    const double nu2,           // the second parameter of distribution (number of de  
    int& error_code             // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of Fisher's F-distribution with the *nu1* and *nu2* parameters. In case of error it returns false. Analog of the [rf\(\)](#) in R.

```
bool MathRandomF(  
    const double nu1,           // the first parameter of distribution (number of de  
    const double nu2,           // the second parameter of distribution (number of de  
    const int data_count,       // amount of required data  
    double& result[]           // array with values of pseudorandom variables  
);
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsF

Calculates the theoretical numerical values of the first 4 moments of the Fisher's F-distribution with the *nu1* and *nu2* parameters.

```
double MathMomentsF(  
    const double  nu1,           // the first parameter of distribution (number of de  
    const double  nu2,           // the second parameter of distribution (number of de  
    double&       mean,          // variable for the mean  
    double&       variance,      // variable for the variance  
    double&       skewness,     // variable for the skewness  
    double&       kurtosis,     // variable for the kurtosis  
    int&          error_code    // variable for the error code  
);
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

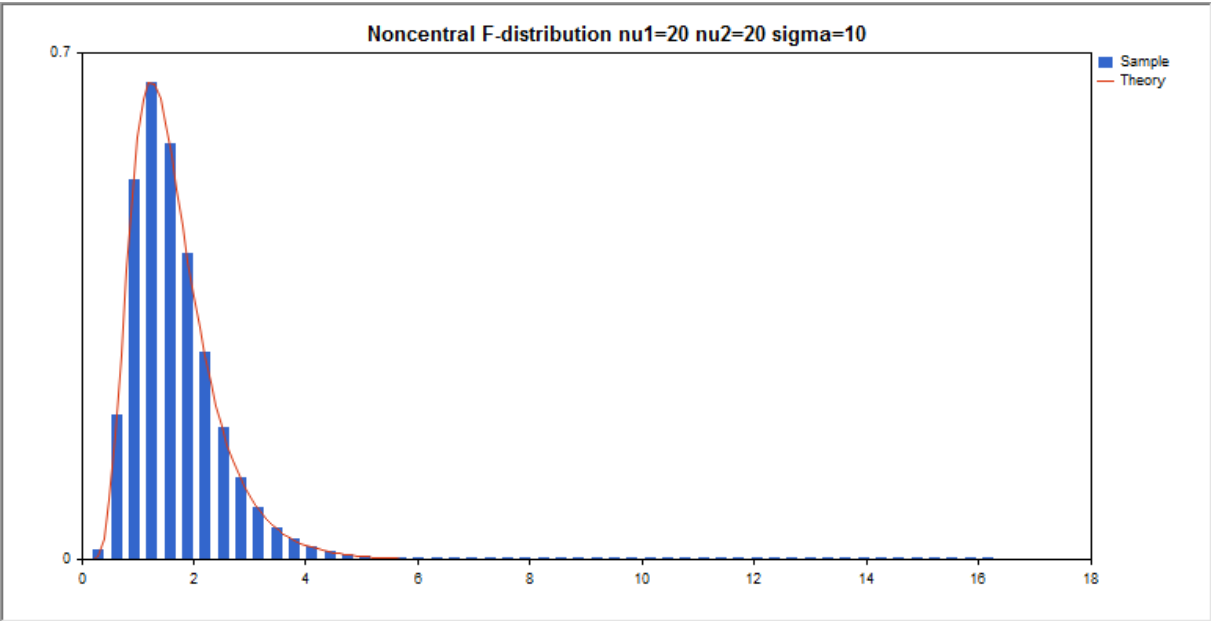
Noncentral F-distribution

This section contains functions for working with noncentral F-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of noncentral Fisher's F-distribution. The noncentral F-distribution is defined by the following formula:

$$f_{NoncentralF}(x|v_1,v_2,\sigma) = e^{-\frac{\sigma}{2}} \sum_{r=0}^{\infty} \frac{1}{r!} \left(\frac{\sigma}{2}\right)^r \frac{\Gamma\left(\frac{v_1+v_2}{2}+r\right)}{\Gamma\left(\frac{v_2}{2}+r\right)\Gamma\left(\frac{v_2}{2}\right)} \left(\frac{v_1}{v_2}\right)^{\frac{v_2}{2}+r} \frac{x^{\frac{v_2}{2}-1+r}}{\left(1+\frac{v_1}{v_2}x\right)^{\frac{v_1+v_2}{2}+r}}$$

where:

- x – value of the random variable
- v1 – the first parameter of distribution (number of degrees of freedom)
- v2 – the second parameter of distribution (number of degrees of freedom)
- σ – noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralF	Calculates the probability density function of the noncentral F-distribution
MathCumulativeDistributionNoncentralF	Calculates the value of the noncentral F-distribution function
MathQuantileNoncentralF	Calculates the value of the inverse noncentral F-distribution function for the specified probability

<u>MathRandomNoncentralF</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the noncentral Fisher's distribution law
<u>MathMomentsNoncentralF</u>	Calculates the theoretical numerical values of the first 4 moments of the noncentral Fisher's F-distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralF.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=20;    // the first number of degrees of freedom
input double nu_2=20;    // the second number of degrees of freedom
input double sig=10;     // noncentrality parameter
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral Fisher's F-distribution
    MathRandomNoncentralF(nu_1,nu_2,sig,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralF(x2,nu_1,nu_2,sig,false,y2);
```



```

//--- set the scale
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral F-distribution nu1=%G nu2=%G sigma="
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```



```
        frequency[ind]++;
    }
    return (true);
}

//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    //-- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //-- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    //-- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityNoncentralF

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [df\(\)](#) in R.

```
bool MathProbabilityDensityNoncentralF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNoncentralF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralF

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability of exceeding the value is calculated
    const bool log_mode,      // flag to calculate the logarithm of the value, if true, then the logarithm is calculated
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralF(
    const double x,           // value of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pf\(\)](#) in R.

```
bool MathCumulativeDistributionNoncentralF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability of exceeding the value is calculated
    const bool log_mode,      // flag to calculate the logarithm of the value, if true, then the logarithm is calculated
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionNoncentralF(
    const double& x[],        // array with the values of random variable
    const double nu1,         // the first parameter of distribution (number of degrees of freedom)
    const double nu2,         // the second parameter of distribution (number of degrees of freedom)
    const double sigma,       // noncentrality parameter
    double& result[]          // array for values of the probability function
);
```



```
);
```

Parameters*x*

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileF

For the specified *probability*, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileF(
    const double probability, // probability value of random variable occurrence
    const double nu1,        // the first parameter of distribution (number of de
    const double nu2,        // the second parameter of distribution (number of de
    const double sigma,      // noncentrality parameter
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileF(
    const double probability, // probability value of random variable occurrence
    const double nu1,        // the first parameter of distribution (number of de
    const double nu2,        // the second parameter of distribution (number of de
    const double sigma,      // noncentrality parameter
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns false. Analog of the [qf\(\)](#) in R.

```
double MathQuantileF(
    const double& probability[], // array with probability values of random variable
    const double nu1,           // the first parameter of distribution (number of de
    const double nu2,           // the second parameter of distribution (number of de
    const double sigma,         // noncentrality parameter
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns false.

```
bool MathQuantileF(
    const double& probability[], // array with probability values of random variable
    const double nu1,           // the first parameter of distribution (number of de
    const double nu2,           // the second parameter of distribution (number of de
    double& result[]            // array with values of quantiles
);
```


Parameters*probability*

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralF

Generates a pseudorandom variable distributed according to the law of noncentral Fisher's F-distribution with the *nu1*, *nu2* and *sigma* parameters. In case of error it returns [NaN](#).

```
double MathRandomNoncentralF(  
    const double nu1,           // the first parameter of distribution (number of de  
    const double nu2,           // the second parameter of distribution (number of de  
    const double sigma,         // noncentrality parameter  
    int& error_code              // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of noncentral Fisher's F-distribution with the *nu1*, *nu2* and *sigma* parameters. In case of error it returns false. Analog of the [rf\(\)](#) in R.

```
bool MathRandomNoncentralF(  
    const double nu1,           // the first parameter of distribution (number of de  
    const double nu2,           // the second parameter of distribution (number of de  
    const double sigma,         // noncentrality parameter  
    const int data_count,       // amount of required data  
    double& result[]            // array with values of pseudorandom variables  
);
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralF

Calculates the theoretical numerical values of the first 4 moments of the noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters.

```
double MathMomentsNoncentralF(  
    const double  nu1,           // the first parameter of distribution (number of de  
    const double  nu2,           // the second parameter of distribution (number of de  
    const double  sigma,         // noncentrality parameter  
    double&       mean,          // variable for the mean  
    double&       variance,      // variable for the variance  
    double&       skewness,      // variable for the skewness  
    double&       kurtosis,      // variable for the kurtosis  
    int&          error_code     // variable for the error code  
);
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

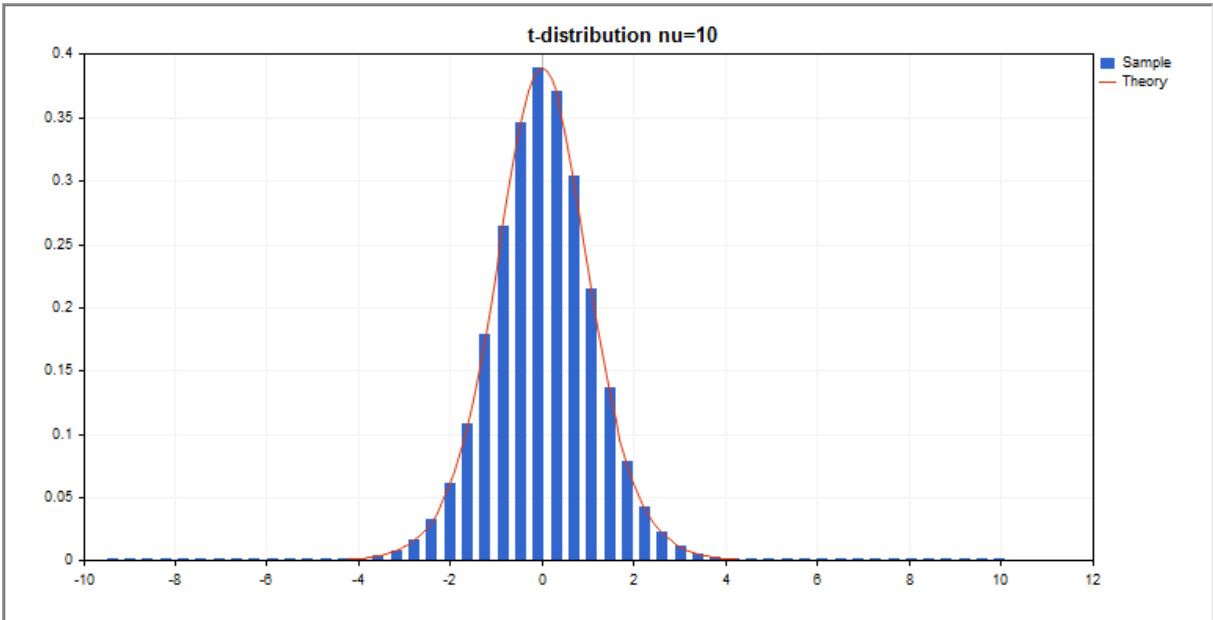
t-distribution

This section contains functions for working with Student's t-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Student's law. The t-distribution is defined by the following formula:

$$f_T(x|\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\pi\nu}} \frac{1}{\left(1+\frac{x^2}{\nu}\right)^{\frac{\nu+1}{2}}}$$

where:

- x — value of the random variable
- ν — parameter of distribution (number of degrees of freedom)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityT	Calculates the probability density function of the t-distribution
MathCumulativeDistributionT	Calculates the value of the t-distribution function
MathQuantileT	Calculates the value of the inverse t-distribution function for the specified probability
MathRandomT	Generates a pseudorandom variable/array of pseudorandom variables distributed according

	to the law of Student's t-distribution
<u>MathMomentsT</u>	Calculates the theoretical numerical values of the first 4 moments of the Student's t-distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\T.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=10;    // the number of degrees of freedom
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;        // the number of values in the sample
    int ncells=51;        // the number of intervals in the histogram
    double x[];           // centers of the histogram intervals
    double y[];           // the number of values from the sample falling within the int
    double data[];        // sample of random values
    double max,min;       // the maximum and minimum values in the sample
//--- obtain a sample from the Student's t-distribution
    MathRandomT(nu_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityT(x2,nu_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;

```



```

    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("t-distribution nu=%G",nu_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```



```
//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityT

Calculates the value of the probability density function of Student's t-distribution with the *nu* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Student's t-distribution with the *nu* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Student's t-distribution with the *nu* parameter for an array of random variables *x[]*. In case of error it returns false. Analog of the [dt\(\)](#) in R.

```
bool MathProbabilityDensityT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of Student's t-distribution with the *nu* parameter for an array of random variables *x[]*. In case of error it returns false.

```
bool MathProbabilityDensityT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionT

Calculates the value of the Student's t-distribution function with the nu parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // flag to calculate the logarithm of the value, if true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Student's t-distribution function with the nu parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Student's t-distribution function with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the [pt\(\)](#) in R.

```
bool MathCumulativeDistributionT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // flag to calculate the logarithm of the value, if true
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the Student's t-distribution function with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileT

For the specified *probability*, the function calculates the value of inverse Student's t-distribution function with the *nu* parameter. In case of error it returns [NaN](#).

```
double MathQuantileT(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse Student's t-distribution function with the *nu* parameter. In case of error it returns [NaN](#).

```
double MathQuantileT(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Student's t-distribution function with the *nu* parameter. In case of error it returns false. Analog of the [qt\(\)](#) in R.

```
double MathQuantileT(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Student's t-distribution function with the *nu* parameter. In case of error it returns false.

```
bool MathQuantileT(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomT

Generates a pseudorandom variable distributed according to the law of Student's t-distribution with the *nu* parameter. In case of error it returns [NaN](#).

```
double MathRandomT(  
    const double nu,           // parameter of distribution (number of degrees of fr  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of Student's t-distribution with the *nu* parameter. In case of error it returns false. Analog of the [rt\(\)](#) in R.

```
bool MathRandomT(  
    const double nu,           // parameter of distribution (number of degrees of fr  
    const int data_count,      // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsT

Calculates the theoretical numerical values of the first 4 moments of the Student's t-distribution with the *nu* parameter.

```
double MathMomentsT(  
    const double  nu,           // parameter of distribution (number of degrees of fr  
    double&       mean,         // variable for the mean  
    double&       variance,     // variable for the variance  
    double&       skewness,     // variable for the skewness  
    double&       kurtosis,     // variable for the kurtosis  
    int&          error_code    // variable for the error code  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

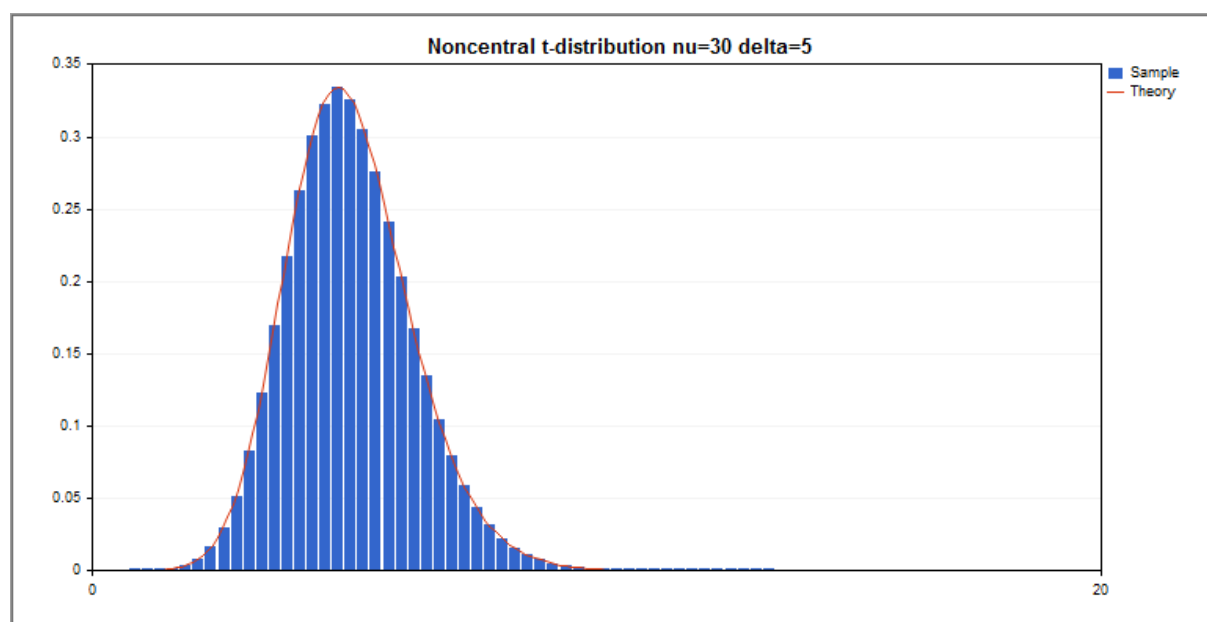
t-distribution

This section contains functions for working with noncentral Student's t-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of noncentral t-distribution. The noncentral t-distribution is defined by the following formula:

$$f_{NoncentralT}(x | \nu, \delta) = \frac{\nu^{\frac{\nu}{2}} e^{-\frac{\delta^2}{2}}}{\Gamma\left(\frac{\nu}{2}\right) \sqrt{\pi} (\nu + x^2)^{\frac{\nu+1}{2}}} \sum_{r=0}^{\infty} \frac{(x\delta)^r}{r!} \left(\frac{2}{\nu + x^2}\right)^{\frac{r}{2}} \Gamma\left(\frac{\nu+r+1}{2}\right)$$

where:

- x — value of the random variable
- ν — parameter of distribution (number of degrees of freedom)
- δ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
<u>MathProbabilityDensityNoncentralT</u>	Calculates the probability density function of the noncentral t-distribution
<u>MathCumulativeDistributionNoncentralT</u>	Calculates the value of the noncentral t-distribution function
<u>MathQuantileNoncentralT</u>	Calculates the value of the inverse noncentral t-distribution function for the specified probability
<u>MathRandomNoncentralT</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according

	to the noncentral Student's t-distribution law
<u>MathMomentsNoncentralT</u>	Calculates the theoretical numerical values of the first 4 moments of the noncentral Student's t-distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralT.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=30;      // the number of degrees of freedom
input double delta_par=5;    // noncentrality parameter
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;      // the number of values in the sample
    int ncells=51;      // the number of intervals in the histogram
    double x[];         // centers of the histogram intervals
    double y[];         // the number of values from the sample falling within the int
    double data[];      // sample of random values
    double max,min;     // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral Student's t-distribution
    MathRandomNoncentralT(nu_par,delta_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralT(x2,nu_par,delta_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];

```



```

double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral t-distribution nu=%G delta=%G",nu,delta));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```



```
    }  
    //+-----+  
    //|  Calculates values for sequence generation  |  
    //+-----+  
    void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)  
    {  
        //--- calculate the absolute range of the sequence to obtain the precision of normalization  
        double range=MathAbs(maxv-minv);  
        int degree=(int)MathRound(MathLog10(range));  
        //--- normalize the maximum and minimum values to the specified precision  
        maxv=NormalizeDouble(maxv,degree);  
        minv=NormalizeDouble(minv,degree);  
        //--- sequence generation step is also set based on the specified precision  
        stepv=NormalizeDouble(MathPow(10,-degree),degree);  
        if ((maxv-minv)/stepv<10)  
            stepv/=10.;  
    }
```


MathProbabilityDensityNoncentralT

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable for the error code
);
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNoncentralT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    int& error_code           // variable for the error code
);
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dt\(\)](#) in R.

```
bool MathProbabilityDensityNoncentralT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityNoncentralT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralT

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // flag to calculate the logarithm of the value, if true
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNoncentralT(
    const double x,           // value of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pt\(\)](#) in R.

```
bool MathCumulativeDistributionNoncentralT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    const bool tail,          // flag of calculation, if true, then the probability of exceeding
    const bool log_mode,      // flag to calculate the logarithm of the value, if true
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionNoncentralT(
    const double& x[],        // array with the values of random variable
    const double nu,          // parameter of distribution (number of degrees of freedom)
    const double delta,       // noncentrality parameter
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralT

For the specified *probability*, the function calculates the value of inverse noncentral Student's t-distribution function with the *nu* and *delta* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralT(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const double delta,      // noncentrality parameter
    const bool tail,         // flag of calculation, if lower_tail=false, then cal
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse noncentral Student's t-distribution function with the *nu* and *delta* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNoncentralT(
    const double probability, // probability value of random variable occurrence
    const double nu,         // parameter of distribution (number of degrees of fr
    const double delta,      // noncentrality parameter
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Student's t-distribution function with the *nu* and *delta* parameters. In case of error it returns false. Analog of the [qt\(\)](#) in R.

```
double MathQuantileNoncentralT(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const double delta,         // noncentrality parameter
    const bool tail,           // flag of calculation, if lower_tail=false, then cal
    const bool log_mode,       // flag of calculation, if log_mode=true, calculation
    double& result[]           // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Student's t-distribution function with the *nu* and *delta* parameters. In case of error it returns false.

```
bool MathQuantileNoncentralT(
    const double& probability[], // array with probability values of random variable
    const double nu,            // parameter of distribution (number of degrees of fr
    const double delta,         // noncentrality parameter
    double& result[]           // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralT

Generates a pseudorandom variable distributed according to the law of noncentral Student's t-distribution with the *nu* and *delta* parameters. In case of error it returns [NaN](#).

```
double MathRandomNoncentralT(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double delta,        // noncentrality parameter  
    int& error_code            // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of noncentral Student's t-distribution with the *nu* and *delta* parameters. In case of error it returns false. Analog of the [rt\(\)](#) in R.

```
bool MathRandomNoncentralT(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double delta,        // noncentrality parameter  
    const int data_count,      // amount of required data  
    double& result[]           // array with values of pseudorandom variables  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralT

Calculates the theoretical numerical values of the first 4 moments of the noncentral Student's t-distribution with the nu and delta parameters.

```
double MathMomentsNoncentralT(  
    const double nu,           // parameter of distribution (number of degrees of freedom)  
    const double delta,        // noncentrality parameter  
    double& mean,              // variable for the mean  
    double& variance,          // variable for the variance  
    double& skewness,          // variable for the skewness  
    double& kurtosis,          // variable for the kurtosis  
    int& error_code            // variable for the error code  
);
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

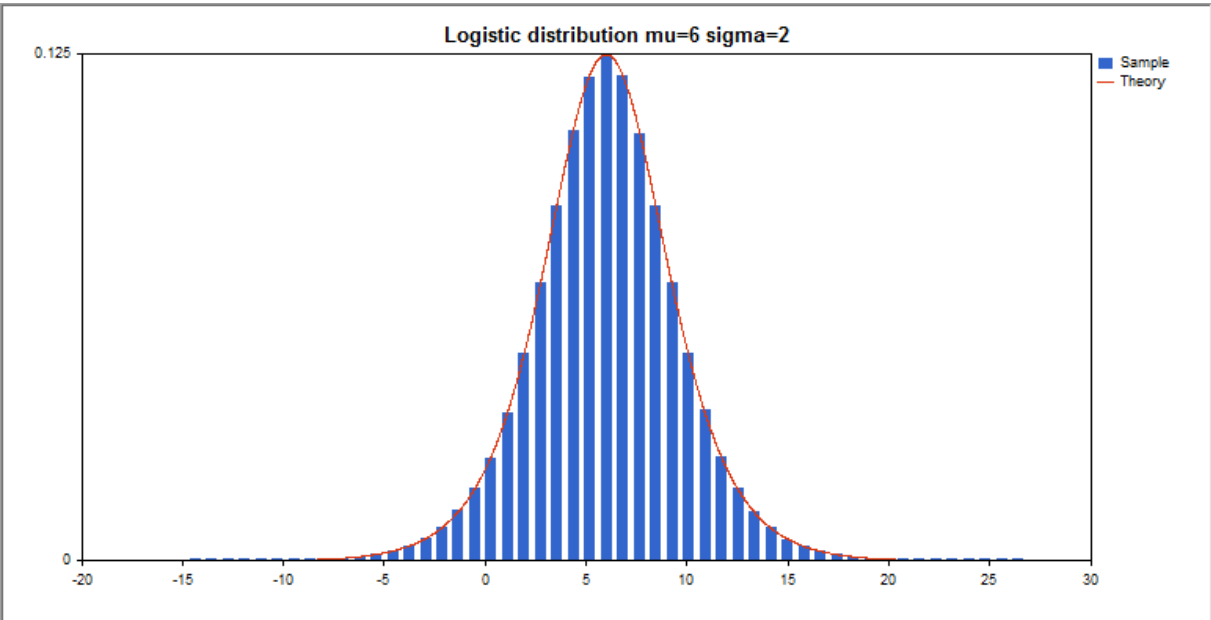
Logistic distribution

This section contains functions for working with logistic distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the logistic law. The logistic distribution is defined by the following formula:

$$f_{Logistic}(x|\mu,\sigma)=\frac{e^{-\frac{x-\mu}{\sigma}}}{\sigma\left(1+e^{-\frac{x-\mu}{\sigma}}\right)^2}$$

where:

- x — value of the random variable
- μ — mean parameter of the distribution
- σ — scale parameter of the distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityLogistic	Calculates the probability density function of the logistic distribution
MathCumulativeDistributionLogistic	Calculates the value of the logistic probability distribution function
MathQuantileLogistic	Calculates the value of the inverse logistic distribution function for the specified probability
MathRandomLogistic	Generates a pseudorandom variable/array of pseudorandom variables distributed according

	to the logistic law
<u>MathMomentsLogistic</u>	Calculates the theoretical numerical values of the first 4 moments of the logistic distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\Logistic.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=6;          // mean parameter of the distribution
input double sigma_par=2;       // scale parameter of the distribution
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0, CHART_SHOW, false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // the number of values in the sample
    int ncells=51;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the int
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
//--- obtain a sample from the logistic distribution
    MathRandomLogistic(mu_par, sigma_par, n, data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data, x, y, max, min, ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max, min, step);
    step=MathMin(step, (max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min, max, step, x2);
    MathProbabilityDensityLogistic(x2, mu_par, sigma_par, false, y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;

```



```

    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Logistic distribution mu=%G sigma=%G",mu_par,sigma_par));
graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(theor_max);
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```



```
        frequency[ind]++;
    }
    return (true);
}

//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityLogistic

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityLogistic(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityLogistic(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dlogis\(\)](#) in R.

```
bool MathProbabilityDensityLogistic(
    const double& x[],         // array with the values of random variable
    const double mu,           // mean parameter of the distribution
    const double sigma,        // scale parameter of the distribution
    const bool log_mode,       // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]           // array for values of the probability density function
);
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityLogistic(
    const double& x[],         // array with the values of random variable
    const double mu,           // mean parameter of the distribution
    const double sigma,        // scale parameter of the distribution
    double& result[]           // array for values of the probability density function
);
```

Parameters

x
[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionLogistic

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionLogistic(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionLogistic(
    const double x,           // value of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    int& error_code           // variable to store the error code
);
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionLogistic(
    const double& x[],        // array with the values of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the [plogis\(\)](#) in R.

```
bool MathCumulativeDistributionLogistic(
    const double& x[],        // array with the values of random variable
    const double mu,          // mean parameter of the distribution
    const double sigma,       // scale parameter of the distribution
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileLogistic

For the specified *probability*, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileLogistic(
    const double probability, // probability value of random variable occurrence
    const double mu,         // mean parameter of the distribution
    const double sigma,      // scale parameter of the distribution
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns [NaN](#).

```
double MathQuantileLogistic(
    const double probability, // probability value of random variable occurrence
    const double mu,         // mean parameter of the distribution
    const double sigma,      // scale parameter of the distribution
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the [qlogis\(\)](#) in R.

```
double MathQuantileLogistic(
    const double& probability[], // array with probability values of random variable
    const double mu,            // mean parameter of the distribution
    const double sigma,         // scale parameter of the distribution
    const bool tail,           // flag of calculation, if false, then calculation is
    const bool log_mode,       // flag of calculation, if log_mode=true, calculation
    double& result[]           // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool MathQuantileLogistic(
    const double& probability[], // array with probability values of random variable
    const double mu,            // mean parameter of the distribution
    const double sigma,         // scale parameter of the distribution
    double& result[]           // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomLogistic

Generates a pseudorandom variable distributed according to the law of logistic distribution with the μ and σ parameters. In case of error it returns [NaN](#).

```
double MathRandomLogistic(  
    const double mu,           // mean parameter of the distribution  
    const double sigma,        // scale parameter of the distribution  
    int& error_code             // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of logistic distribution with the μ and σ parameters. In case of error it returns false. Analog of the [rlogis\(\)](#) in R.

```
bool MathRandomLogistic(  
    const double mu,           // mean parameter of the distribution  
    const double sigma,        // scale parameter of the distribution  
    const int data_count,      // amount of required data  
    double& result[]           // array with values of pseudorandom variables  
);
```

Parameters

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsLogistic

Calculates the theoretical numerical values of the first 4 moments of the logistic distribution with the μ and σ parameters.

```
double MathMomentsLogistic(  
    const double mu,           // mean parameter of the distribution  
    const double sigma,       // scale parameter of the distribution  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

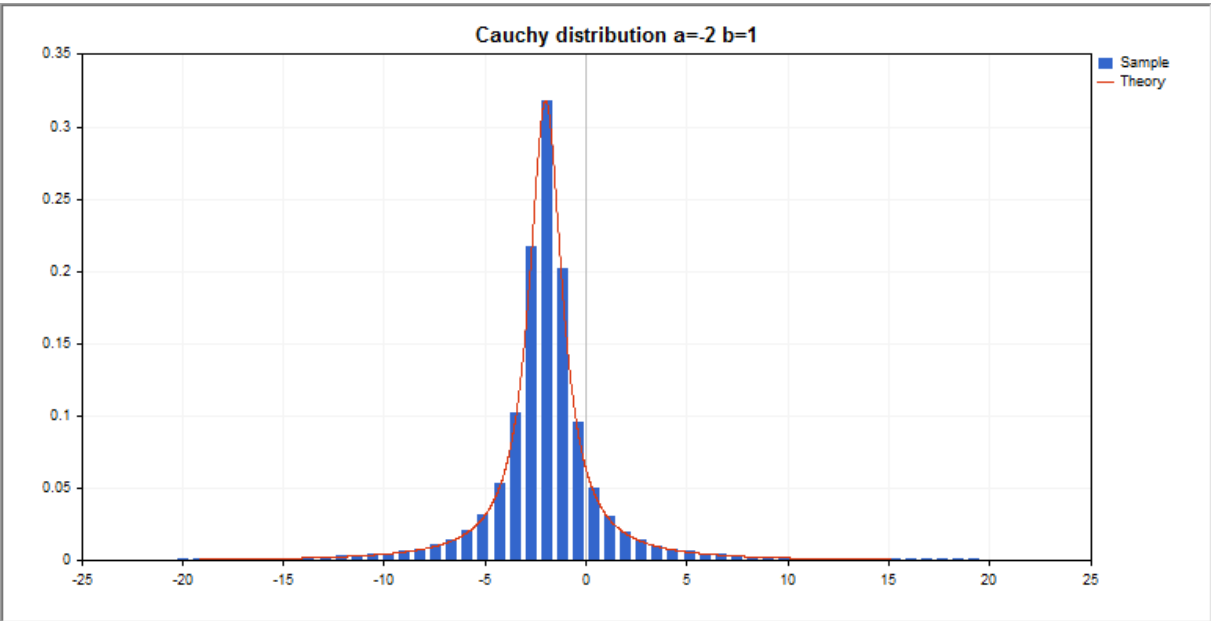
Cauchy distribution

This section contains functions for working with Cauchy distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Cauchy law. The Cauchy distribution is defined by the following formula:

$$f_{Cauchy}(x|a,b) = \frac{1}{\pi} \frac{b}{b^2 + (x-a)^2}$$

where:

- x – value of the random variable
- a – mean parameter of the distribution
- b – scale parameter of the distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityCauchy	Calculates the probability density function of the Cauchy distribution
MathCumulativeDistributionCauchy	Calculates the value of the Cauchy probability distribution function
MathQuantileCauchy	Calculates the value of the inverse Cauchy distribution function for the specified probability
MathRandomCauchy	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Cauchy law

MathMomentsCauchy

Calculates the theoretical numerical values of the first 4 moments of the Cauchy distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Cauchy.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=-2;      // mean parameter of the distribution
input double b_par=1;      // scale parameter of the distribution
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=51;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the int
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
//--- obtain a sample from the Cauchy distribution
    MathRandomCauchy(a_par,b_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityCauchy(x2,a_par,b_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Cauchy distribution a=%G b=%G",a_par,b_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1)
        return(false);
    int size=ArraySize(data);
    if(size<cells*10)
        return(false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    Print("min=",minv," max=",maxv);
    minv=-20;
    maxv=20;
    double range=maxv-minv;
    double width=range/cells;
    if(width==0)
        return(false);
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=(int)MathRound((data[i]-minv)/width);

```



```
        if(ind>=0 && ind<cells)
            frequency[ind]++;
    }
    return(true);
}

//+-----+
//|  Calculates values for sequence generation          |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityCauchy

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityCauchy(
    const double x,           // value of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityCauchy(
    const double x,           // value of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dcauchy\(\)](#) in R.

```
bool MathProbabilityDensityCauchy(
    const double& x[],        // array with the values of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityCauchy(
    const double& x[],        // array with the values of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    double& result[]          // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionCauchy

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionCauchy(
    const double x,           // value of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionCauchy(
    const double x,           // value of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionCauchy(
    const double& x[],        // array with the values of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [plogis\(\)](#) in R.

```
bool MathCumulativeDistributionCauchy(
    const double& x[],        // array with the values of random variable
    const double a,           // mean parameter of the distribution
    const double b,           // scale parameter of the distribution
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileCauchy

For the specified *probability*, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileCauchy(
    const double probability, // probability value of random variable occurrence
    const double a,          // mean parameter of the distribution
    const double b,          // scale parameter of the distribution
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileCauchy(
    const double probability, // probability value of random variable occurrence
    const double a,          // mean parameter of the distribution
    const double b,          // scale parameter of the distribution
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns false. Analog of the [qcauchy\(\)](#) in R.

```
double MathQuantileCauchy(
    const double& probability[], // array with probability values of random variable
    const double a,              // mean parameter of the distribution
    const double b,              // scale parameter of the distribution
    const bool tail,             // flag of calculation, if false, then calculation is
    const bool log_mode,         // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns false.

```
bool MathQuantileCauchy(
    const double& probability[], // array with probability values of random variable
    const double a,              // mean parameter of the distribution
    const double b,              // scale parameter of the distribution
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomCauchy

Generates a pseudorandom variable distributed according to the law of Cauchy distribution with the *a* and *b* parameters. In case of error it returns [NaN](#).

```
double MathRandomCauchy(  
    const double a,           // mean parameter of the distribution  
    const double b,           // scale parameter of the distribution  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of Cauchy distribution with the *a* and *b* parameters. In case of error it returns false. Analog of the [rcauchy\(\)](#) in R.

```
bool MathRandomCauchy(  
    const double a,           // mean parameter of the distribution  
    const double b,           // scale parameter of the distribution  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsCauchy

Calculates the theoretical numerical values of the first 4 moments of the Cauchy distribution with the *a* and *b* parameters.

```
double MathMomentsCauchy(  
    const double a,           // mean parameter of the distribution  
    const double b,           // scale parameter of the distribution  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a
[in] mean parameter of the distribution.

b
[in] scale parameter of the distribution.

mean
[out] Variable to get the mean value.

variance
[out] Variable to get the variance.

skewness
[out] Variable to get the skewness.

kurtosis
[out] Variable to get the kurtosis.

error_code
[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

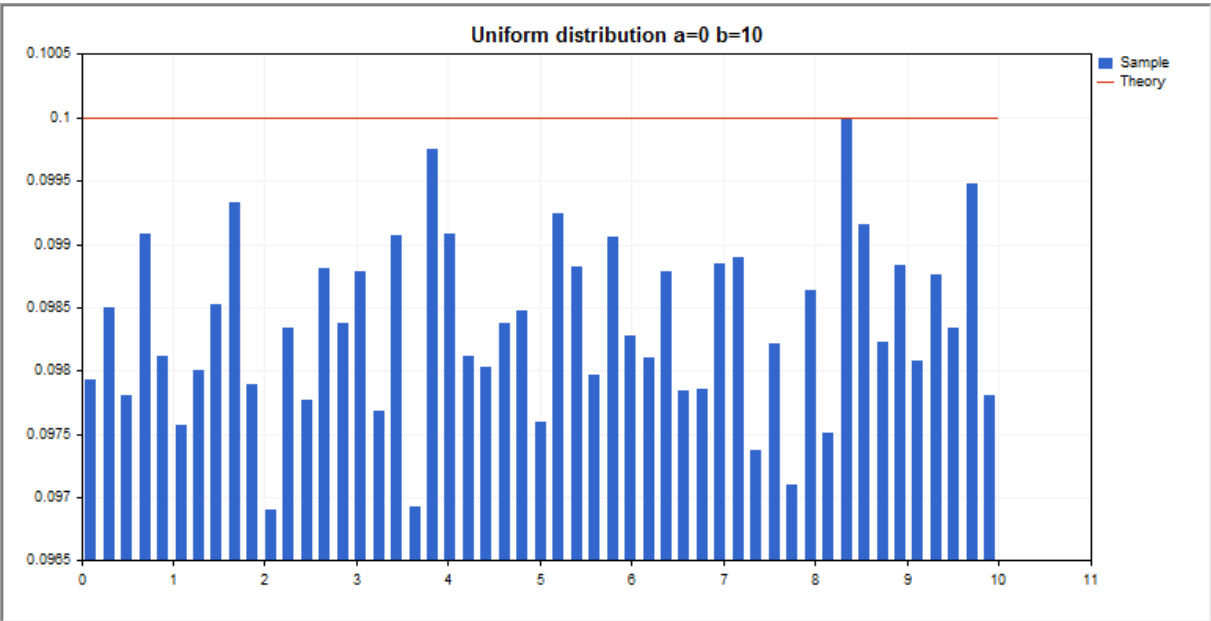
Uniform distribution

This section contains functions for working with uniform distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the uniform law. The uniform distribution is defined by the following formula:

$$f_{Uniform}(x|a,b)=\frac{1}{b-a}$$

where:

- x – value of the random variable
- a – parameter of the distribution (lower bound)
- b – parameter of the distribution (upper bound)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityUniform	Calculates the probability density function of the uniform distribution
MathCumulativeDistributionUniform	Calculates the value of the uniform probability distribution function
MathQuantileUniform	Calculates the value of the inverse uniform distribution function for the specified probability
MathRandomUniform	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the uniform law

MathMomentsUniform

Calculates the theoretical numerical values of the first 4 moments of the uniform distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Uniform.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=0;      // distribution parameter a (lower bound)
input double b_par=10;     // distribution parameter b (upper bound)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
    //--- initialize the random number generator
    MathSrand(GetTickCount());
    //--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=51;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the int
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
    //--- obtain a sample from the uniform distribution
    MathRandomUniform(a_par,b_par,n,data);
    //--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
    //--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityUniform(x2,a_par,b_par,false,y2);
    //--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
        CGraphic graphic;
        if(ObjectFind(chart,name)<0)
            graphic.Create(chart,name,0,0,0,780,380);
        else
            graphic.Attach(chart,name);
        graphic.BackgroundMain(StringFormat("Uniform distribution a=%G b=%G",a_par,b_par));
        graphic.BackgroundMainSize(16);
//--- plot all curves
        graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
        graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
        graphic.CurvePlotAll();
//--- plot all curves
        graphic.Update();
    }
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+

```



```
///| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv, double &minv, double &stepv)
{
//--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv, degree);
    minv=NormalizeDouble(minv, degree);
//--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10, -degree), degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityUniform

Calculates the value of the probability density function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityUniform(
    const double x,           // value of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityUniform(
    const double x,           // value of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dunif\(\)](#) in R.

```
bool MathProbabilityDensityUniform(
    const double& x[],        // array with the values of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityUniform(
    const double& x[],        // array with the values of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionUniform

Calculates the probability distribution function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionUniform(
    const double x,           // value of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionUniform(
    const double x,           // value of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    int& error_code           // variable to store the error code
);
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionUniform(
    const double& x[],        // array with the values of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [punif\(\)](#) in R.

```
bool MathCumulativeDistributionUniform(
    const double& x[],        // array with the values of random variable
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileUniform

For the specified *probability*, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileUniform(
    const double probability, // probability value of random variable occurrence
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    const bool tail,          // flag of calculation, if false, then calculation is
    const bool log_mode,      // flag of calculation, if log_mode=true, calculation
    int& error_code           // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileUniform(
    const double probability, // probability value of random variable occurrence
    const double a,           // distribution parameter a (lower bound)
    const double b,           // distribution parameter b (upper bound)
    int& error_code           // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns false. Analog of the [qcauschy\(\)](#) in R.

```
double MathQuantileUniform(
    const double& probability[], // array with probability values of random variable
    const double a,              // distribution parameter a (lower bound)
    const double b,              // distribution parameter b (upper bound)
    const bool tail,             // flag of calculation, if false, then calculation is
    const bool log_mode,         // flag of calculation, if log_mode=true, calculation
    double& result[]             // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns false.

```
bool MathQuantileUniform(
    const double& probability[], // array with probability values of random variable
    const double a,              // distribution parameter a (lower bound)
    const double b,              // distribution parameter b (upper bound)
    double& result[]             // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomUniform

Generates a pseudorandom variable distributed according to the law of uniform distribution with the *a* and *b* parameters. In case of error it returns [NaN](#).

```
double MathRandomUniform(  
    const double a,           // distribution parameter a (lower bound)  
    const double b,           // distribution parameter b (upper bound)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of uniform distribution with the *a* and *b* parameters. In case of error it returns false. Analog of the [runif\(\)](#) in R.

```
bool MathRandomUniform(  
    const double a,           // distribution parameter a (lower bound)  
    const double b,           // distribution parameter b (upper bound)  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

a

[in] Distribution parameter *a* (lower bound).

b

[in] Distribution parameter *b* (upper bound).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsUniform

Calculates the theoretical numerical values of the first 4 moments of the uniform distribution with the a and b parameters.

```
double MathMomentsUniform(  
    const double a,           // distribution parameter a (lower bound)  
    const double b,           // distribution parameter b (upper bound)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

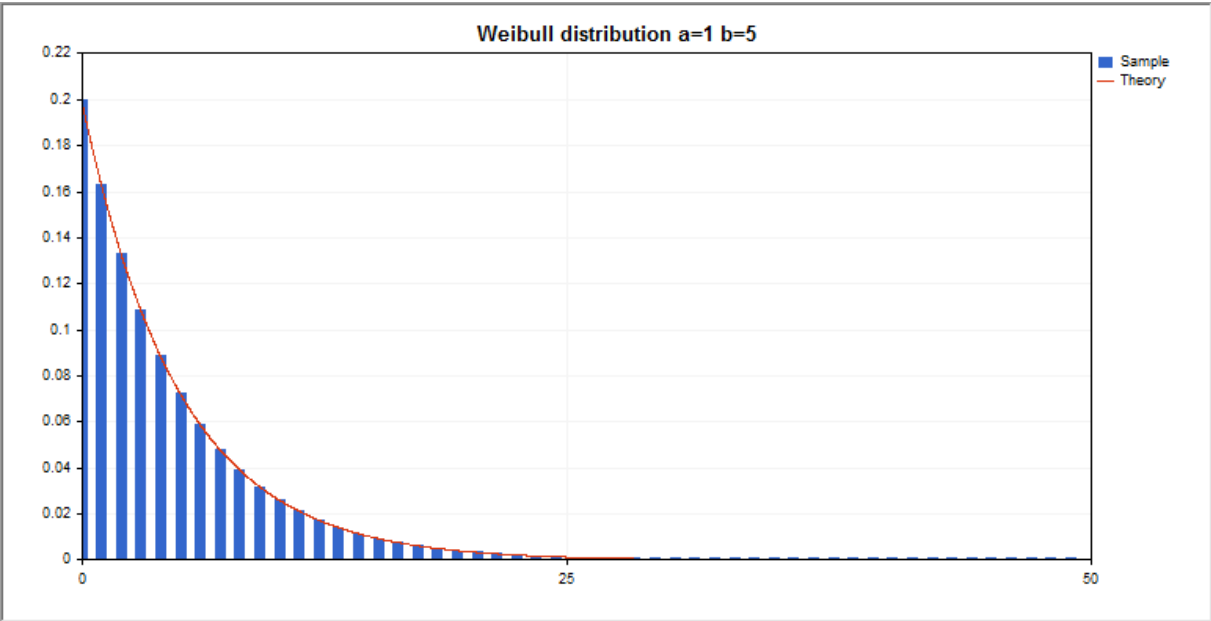
Weibull distribution

This section contains functions for working with Weibull distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Weibull law. The Weibull distribution is defined by the following formula:

$$f_{\text{Weibull}}\left(x|a,b\right)=\frac{a}{b}\left(\frac{x}{b}\right)^{a-1}e^{-\left(\frac{x}{b}\right)^a}$$

where:

- x – value of the random variable
- a – parameter of the distribution (shape)
- b – parameter of the distribution (scale)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityWeibull	Calculates the probability density function of the Weibull distribution
MathCumulativeDistributionWeibull	Calculates the value of the Weibull probability distribution function
MathQuantileWeibull	Calculates the value of the inverse Weibull distribution function for the specified probability
MathRandomWeibull	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Weibull law

MathMomentsWeibull

Calculates the theoretical numerical values of the first 4 moments of the Weibull distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Weibull.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=1;      // parameter of the distribution (shape)
input double b_par=5;      // parameter of the distribution (scale)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
    //--- initialize the random number generator
    MathSrand(GetTickCount());
    //--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=51;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the intervals
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
    //--- obtain a sample from the Weibull distribution
    MathRandomWeibull(a_par,b_par,n,data);
    //--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
    //--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityWeibull(x2,a_par,b_par,false,y2);
    //--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
```



```

        y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Weibull distribution a=%G b=%G",a_par,b_par));
graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
graphic.XAxis().AutoScale(false);
graphic.XAxis().Max(max);
graphic.XAxis().Min(min);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```



```
    }
    return (true);
}
//+-----+
//|  Calculates values for sequence generation  |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //--- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //--- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //--- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityWeibull

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityWeibull(
    const double x,           // value of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityWeibull(
    const double x,           // value of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [dweibull\(\)](#) in R.

```
bool MathProbabilityDensityWeibull(
    const double& x[],        // array with the values of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityWeibull(
    const double& x[],        // array with the values of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionWeibull

Calculates the value of the Weibull distribution function with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionWeibull(
    const double x,           // value of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Weibull distribution function with the a and b parameters for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionWeibull(
    const double x,           // value of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Weibull distribution function with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the [pweibull\(\)](#) in R.

```
bool MathCumulativeDistributionWeibull(
    const double& x[],        // array with the values of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the Weibull distribution function with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionWeibull(
    const double& x[],        // array with the values of random variable
    const double a,           // parameter of the distribution (shape)
    const double b,           // parameter of the distribution (scale)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileWeibull

For the specified *probability*, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileWeibull(
    const double probability, // probability value of random variable occurrence
    const double a,          // parameter of the distribution (shape)
    const double b,          // parameter of the distribution (scale)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns [NaN](#).

```
double MathQuantileWeibull(
    const double probability, // probability value of random variable occurrence
    const double a,          // parameter of the distribution (shape)
    const double b,          // parameter of the distribution (scale)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns false. Analog of the [qweibull\(\)](#) in R.

```
double MathQuantileWeibull(
    const double& probability[], // array with probability values of random variable
    const double a,             // parameter of the distribution (shape)
    const double b,             // parameter of the distribution (scale)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns false.

```
bool MathQuantileWeibull(
    const double& probability[], // array with probability values of random variable
    const double a,             // parameter of the distribution (shape)
    const double b,             // parameter of the distribution (scale)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomWeibull

Generates a pseudorandom variable distributed according to the law of Weibull distribution with the *a* and *b* parameters. In case of error it returns [NaN](#).

```
double MathRandomWeibull(  
    const double a,           // parameter of the distribution (shape)  
    const double b,           // parameter of the distribution (scale)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of Weibull distribution with the *a* and *b* parameters. In case of error it returns false. Analog of the [rweibull\(\)](#) in R.

```
bool MathRandomWeibull(  
    const double a,           // parameter of the distribution (shape)  
    const double b,           // parameter of the distribution (scale)  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsWeibull

Calculates the theoretical numerical values of the first 4 moments of the Weibull distribution with the *a* and *b* parameters.

```
double MathMomentsWeibull(  
    const double a,           // parameter of the distribution (shape)  
    const double b,           // parameter of the distribution (scale)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

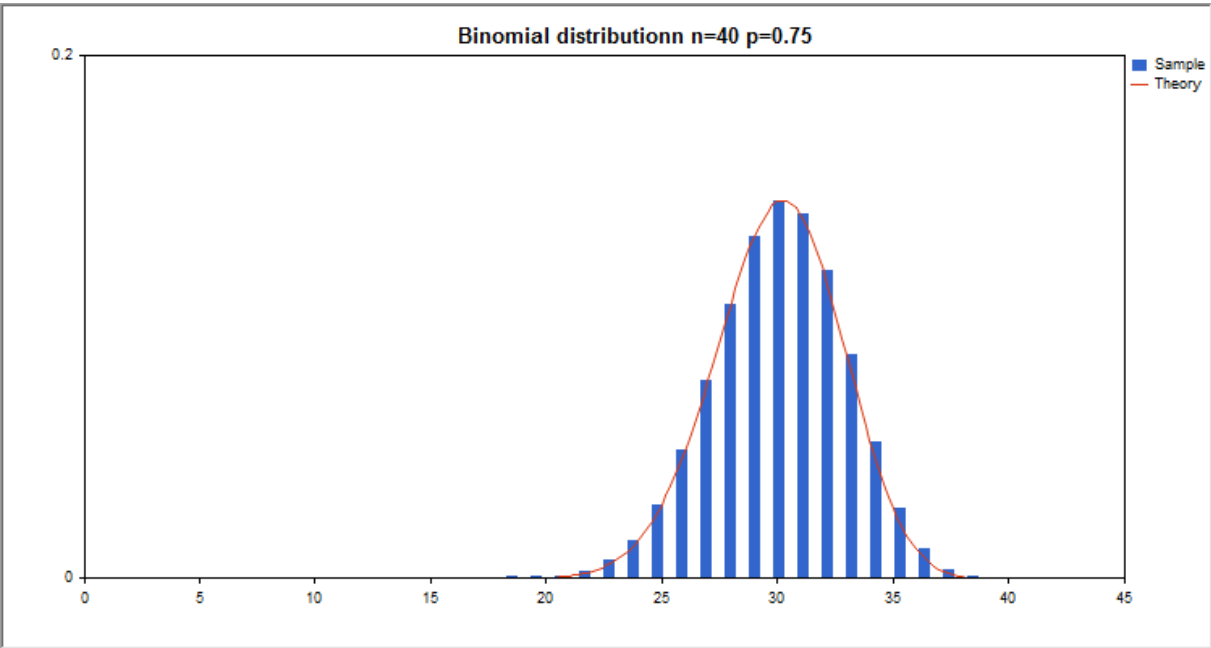
Binomial distribution

This section contains functions for working with binomial distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the binomial law. The binomial distribution is defined by the following formula:

$$f_{\text{Binomial}}(x|n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$

where:

- x – value of the random variable
- n – number of tests
- p – probability of success for each test



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityBinomial	Calculates the probability density function of the binomial distribution
MathCumulativeDistributionBinomial	Calculates the value of the binomial probability distribution function
MathQuantileBinomial	Calculates the value of the inverse binomial distribution function for the specified probability
MathRandomBinomial	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the binomial law

MathMomentsBinomial

Calculates the theoretical numerical values of the first 4 moments of the binomial distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Binomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;          // the number of tests
input double p_par=0.75;        // probability of success for each test
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=20;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the interval
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
//--- obtain a sample from the binomial distribution
    MathRandomBinomial(n_par,p_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(0,n_par,1,x2);
    MathProbabilityDensityBinomial(x2,n_par,p_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
```



```

        graphic.Create(chart,name,0,0,0,780,380);
    else
        graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("Binomial distributionn n=%G p=%G",n_par,p_par));
    graphic.BackgroundMainSize(16);
    //--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
    //--- and now plot the theoretical curve of the distribution density
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
    graphic.CurvePlotAll();
    //--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```


MathProbabilityDensityBinomial

Calculates the value of the probability mass function of binomial distribution with the *n* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityBinomial(
    const double x,           // value of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of binomial distribution with the *n* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityBinomial(
    const double x,           // value of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of binomial distribution with the *n* and *p* parameters for an array of random variables *x[]*. In case of error it returns false. Analog of the [dbinom\(\)](#) in R.

```
bool MathProbabilityDensityBinomial(
    const double& x[],        // array with the values of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability mass function of binomial distribution with the *n* and *p* parameters for an array of random variables *x[]*. In case of error it returns false.

```
bool MathProbabilityDensityBinomial(
    const double& x[],        // array with the values of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionBinomial

Calculates the value of the probability distribution function for binomial law with the n and p parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathCumulativeDistributionBinomial(
    const double x,           // value of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathCumulativeDistributionBinomial(
    const double x,           // value of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for an array of random variables $x[]$. In case of error it returns false. Analog of the [pweibull\(\)](#) in R.

```
bool MathCumulativeDistributionBinomial(
    const double& x[],        // array with the values of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for an array of random variables $x[]$. In case of error it returns false.

```
bool MathCumulativeDistributionBinomial(
    const double& x[],        // array with the values of random variable
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileBinomial

For the specified *probability*, the function calculates the inverse value of distribution function for binomial law with the *n* and *p* parameters. In case of error it returns [NaN](#).

```
double MathQuantileBinomial(
    const double probability, // probability value of random variable occurrence
    const double n,          // parameter of the distribution (number of tests)
    const double p,          // parameter of the distribution (probability of event)
    const bool tail,         // flag of calculation, if false, then calculation is for the head
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation is for the log-likelihood
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the inverse value of distribution function for binomial law with the *n* and *p* parameters. In case of error it returns [NaN](#).

```
double MathQuantileBinomial(
    const double probability, // probability value of random variable occurrence
    const double n,          // parameter of the distribution (number of tests)
    const double p,          // parameter of the distribution (probability of event)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for binomial law with the *n* and *p* parameters. In case of error it returns false. Analog of the [qbinom\(\)](#) in R.

```
double MathQuantileBinomial(
    const double& probability[], // array with probability values of random variable
    const double n,             // parameter of the distribution (number of tests)
    const double p,             // parameter of the distribution (probability of event)
    const bool tail,            // flag of calculation, if false, then calculation is for the head
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation is for the log-likelihood
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for binomial law with the *n* and *p* parameters. In case of error it returns false.

```
bool MathQuantileBinomial(
    const double& probability[], // array with probability values of random variable
    const double n,             // parameter of the distribution (number of tests)
    const double p,             // parameter of the distribution (probability of event)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomBinomial

Generates a pseudorandom variable distributed according to the law of binomial distribution with the n and p parameters. In case of error it returns [NaN](#).

```
double MathRandomBinomial (
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    int& error_code           // variable to store the error code
);
```

Generates pseudorandom variables distributed according to the law of binomial distribution with the n and p parameters. In case of error it returns false. Analog of the [rweibull\(\)](#) in R.

```
bool MathRandomBinomial (
    const double n,           // parameter of the distribution (number of tests)
    const double p,           // parameter of the distribution (probability of event)
    const int data_count,     // amount of required data
    double& result[]          // array with values of pseudorandom variables
);
```

Parameters

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

$error_code$

[out] Variable to store the error code.

$data_count$

[out] Amount of required data.

$result[]$

[out] Array to obtain the values of pseudorandom variables.

MathMomentsBinomial

Calculates the theoretical numerical values of the first 4 moments of the binomial distribution with the *n* and *p* parameters.

```
double MathMomentsBinomial(  
    const double n,           // parameter of the distribution (number of tests)  
    const double p,           // parameter of the distribution (probability of event occurrence)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

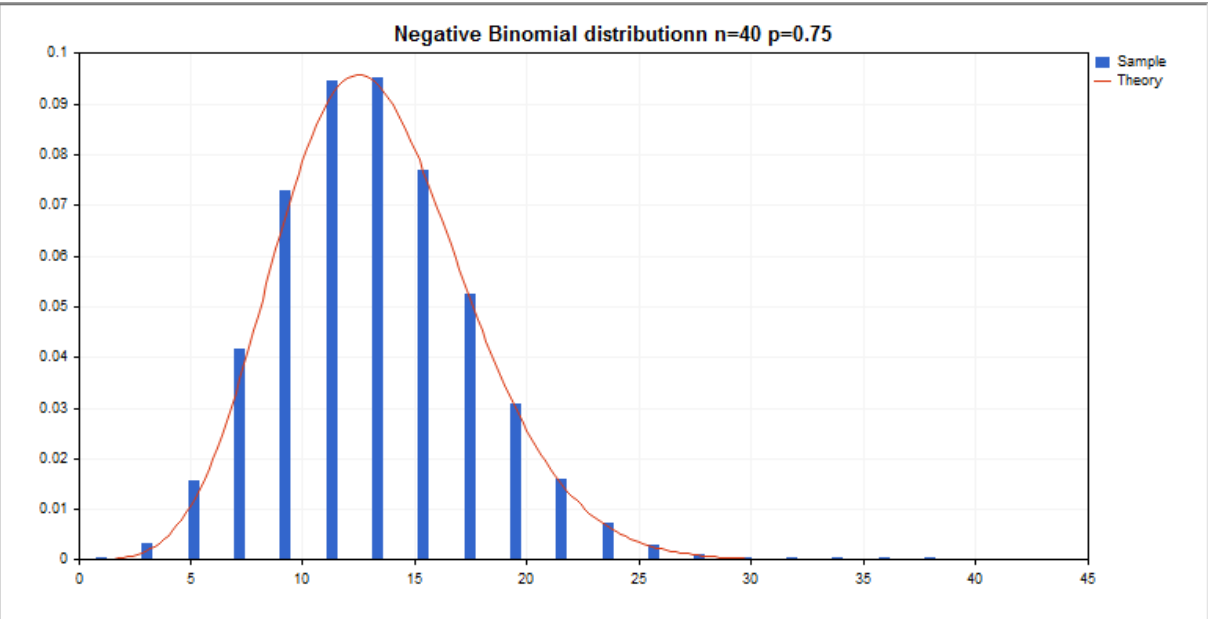
Negative binomial distribution

This section contains functions for working with negative binomial distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the negative binomial law. The negative binomial distribution is defined by the following formula:

$$f_{\text{NegativeBinomial}}(x|r,p) = \frac{\Gamma(r+x)}{\Gamma(r)\Gamma(x+1)} p^r (1-p)^x$$

where:

- x – value of the random variable
- r – number of successful tests
- p – probability of success



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNegativeBinomial	Calculates the probability density function of the negative binomial distribution
MathCumulativeDistributionNegativeBinomial	Calculates the value of the negative binomial probability distribution function
MathQuantileNegativeBinomial	Calculates the value of the inverse negative binomial distribution function for the specified probability
MathRandomNegativeBinomial	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the negative binomial law

MathMomentsNegativeBinomial

Calculates the theoretical numerical values of the first 4 moments of the negative binomial distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NegativeBinomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;           // the number of tests
input double p_par=0.75;         // probability of success for each test
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // the number of values in the sample
    int ncells=19;           // the number of intervals in the histogram
    double x[];              // centers of the histogram intervals
    double y[];              // the number of values from the sample falling within the int
    double data[];           // sample of random values
    double max,min;          // the maximum and minimum values in the sample
//--- obtain a sample from the negative binomial distribution
    MathRandomNegativeBinomial(n_par,p_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(0,n_par,1,x2);
    MathProbabilityDensityNegativeBinomial(x2,n_par,p_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
    CGraphic graphic;
```



```

    if (ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);
    else
        graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("Negative Binomial distributionn n=%G p=%G",n,p));
    graphic.BackgroundMainSize(16);
//--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
    graphic.CurvePlotAll();
//--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```


MathProbabilityDensityNegativeBinomial

Calculates the value of the probability mass function of negative binomial distribution with the *r* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNegativeBinomial(
    const double x,           // value of random variable (integer)
    const double r,           // number of successful tests
    const double p,           // probability of success
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of negative binomial distribution with the *r* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathProbabilityDensityNegativeBinomial(
    const double x,           // value of random variable (integer)
    const double r,           // number of successful tests
    const double p,           // probability of success
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of negative binomial distribution with the *r* and *p* parameters for an array of random variables *x[]*. In case of error it returns false. Analog of the [dnbinom\(\)](#) in R.

```
bool MathProbabilityDensityNegativeBinomial(
    const double& x[],        // array with the values of random variable
    const double r,           // number of successful tests
    const double p,           // probability of success
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability mass function of negative binomial distribution with the *r* and *p* parameters for an array of random variables *x[]*. In case of error it returns false.

```
bool MathProbabilityDensityNegativeBinomial(
    const double& x[],        // array with the values of random variable
    const double r,           // number of successful tests
    const double p,           // probability of success
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

r

[in] Number of successful tests

p

[in] Probability of success.

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNegativeBinomial

Calculates the value of the probability distribution function for negative binomial law with the *r* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNegativeBinomial(
    const double x,           // value of random variable (integer)
    const double r,           // number of successful tests
    const double p,           // probability of success
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for negative binomial law with the *r* and *p* parameters for a random variable *x*. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionNegativeBinomial(
    const double x,           // value of random variable (integer)
    const double r,           // number of successful tests
    const double p,           // probability of success
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for negative binomial law with the *r* and *p* parameters for an array of random variables *x[]*. In case of error it returns false. Analog of the [pweibull\(\)](#) in R.

```
bool MathCumulativeDistributionNegativeBinomial(
    const double& x[],        // array with the values of random variable
    const double r,           // number of successful tests
    const double p,           // probability of success
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]         // array for values of the probability function
);
```

Calculates the value of the probability distribution function for negative binomial law with the *r* and *p* parameters for an array of random variables *x[]*. In case of error it returns false.

```
bool MathCumulativeDistributionNegativeBinomial(
    const double& x[],        // array with the values of random variable
    const double r,           // number of successful tests
    const double p,           // probability of success
    double& result[]         // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

r

[in] Number of successful tests.

p

[in] Probability of success.

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNegativeBinomial

For the specified *probability*, the function calculates the inverse value of distribution function for negative binomial law with the *r* and *p* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNegativeBinomial(
    const double probability, // probability value of random variable occurrence
    const double r,          // number of successful tests
    const double p,          // probability of success
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the inverse value of distribution function for negative binomial law with the *r* and *p* parameters. In case of error it returns [NaN](#).

```
double MathQuantileNegativeBinomial(
    const double probability, // probability value of random variable occurrence
    const double r,          // number of successful tests
    const double p,          // probability of success
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for negative binomial law with the *r* and *p* parameters. In case of error it returns false. Analog of the [qnbinom\(\)](#) in R.

```
double MathQuantileNegativeBinomial(
    const double& probability[], // array with probability values of random variable
    const double r,             // number of successful tests
    const double p,             // probability of success
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for negative binomial law with the *r* and *p* parameters. In case of error it returns false.

```
bool MathQuantileNegativeBinomial(
    const double& probability[], // array with probability values of random variable
    const double r,             // number of successful tests
    const double p,             // probability of success
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

r

[in] Number of successful tests.

p

[in] Probability of success.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNegativeBinomial

Generates a pseudorandom variable distributed according to the law of negative binomial distribution with the *r* and *p* parameters. In case of error it returns [NaN](#).

```
double MathRandomNegativeBinomial(  
    const double r,           // number of successful tests  
    const double p,           // probability of success  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of negative binomial distribution with the *r* and *p* parameters. In case of error it returns false. Analog of the [rweibull\(\)](#) in R.

```
bool MathRandomNegativeBinomial(  
    const double r,           // number of successful tests  
    const double p,           // probability of success  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

r

[in] Number of successful tests.

p

[in] Probability of success.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNegativeBinomial

Calculates the theoretical numerical values of the first 4 moments of the negative binomial distribution with the *r* and *p* parameters.

```
double MathMomentsNegativeBinomial(  
    const double r,           // number of successful tests  
    const double p,           // probability of success  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code          // variable for the error code  
);
```

Parameters

r

[in] Number of successful tests.

p

[in] Probability of success.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

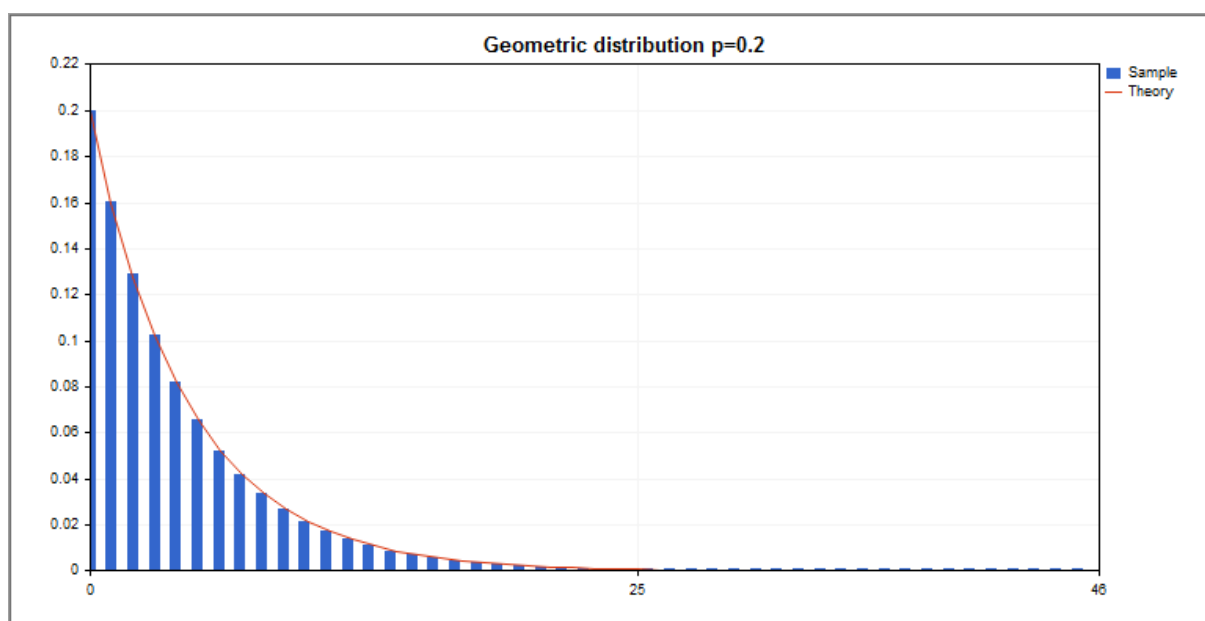
Geometric distribution

This section contains functions for working with geometric distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the geometric law. The geometric distribution is defined by the following formula:

$$f_{\text{Geometric}}(x|p) = p(1-p)^x$$

where:

- x – value of the random variable (integer)
- p – probability of event occurrence in one test



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
<u>MathProbabilityDensityGeometric</u>	Calculates the probability density function of the geometric distribution
<u>MathCumulativeDistributionGeometric</u>	Calculates the value of the geometric probability distribution function
<u>MathQuantileGeometric</u>	Calculates the value of the inverse geometric distribution function for the specified probability
<u>MathRandomGeometric</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the geometric law
<u>MathMomentsGeometric</u>	Calculates the theoretical numerical values of the first 4 moments of the geometric distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\Geometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double p_par=0.2;      // probability of event occurrence in one test
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0, CHART_SHOW, false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;      // the number of values in the sample
    int ncells=47;      // the number of intervals in the histogram
    double x[];         // centers of the histogram intervals
    double y[];         // the number of values from the sample falling within the int
    double data[];      // sample of random values
    double max,min;     // the maximum and minimum values in the sample
//--- obtain a sample from the geometric distribution
    MathRandomGeometric(p_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(0,ncells,1,x2);
    MathProbabilityDensityGeometric(x2,p_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);

```



```

else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Geometric distribution p=%G",p_par));
graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
graphic.XAxis().AutoScale(false);
graphic.XAxis().Max(max);
graphic.XAxis().Min(min);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
    //--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |

```



```
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //-- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //-- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //-- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityGeometric

Calculates the value of the probability mass function of geometric distribution with the p parameter for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityGeometric(
    const double x,           // value of random variable (integer)
    const double p,           // parameter of the distribution (probability of event)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityGeometric(
    const double x,           // value of random variable (integer)
    const double p,           // parameter of the distribution (probability of event)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for an array of random variables $x[]$. In case of error it returns false. Analog of the [dgeom\(\)](#) in R.

```
bool MathProbabilityDensityGeometric(
    const double& x[],        // array with the values of random variable
    const double p,           // parameter of the distribution (probability of event)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]          // array for values of the probability density function
);
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for an array of random variables $x[]$. In case of error it returns false.

```
bool MathProbabilityDensityGeometric(
    const double& x[],        // array with the values of random variable
    const double p,           // parameter of the distribution (probability of event)
    double& result[]          // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

$x[]$

[in] Array with the values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

\log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionGeometric

Calculates the value of the probability distribution function for geometric law with the *p* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionGeometric(
    const double x,           // value of random variable (integer)
    const double p,           // parameter of the distribution (probability of event)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for geometric law with the *p* parameter for a random variable *x*. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionGeometric(
    const double x,           // value of random variable (integer)
    const double p,           // parameter of the distribution (probability of event)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for geometric law with the *p* parameter for an array of random variables *x[]*. In case of error it returns false. Analog of the [pgeom\(\)](#) in R.

```
bool MathCumulativeDistributionGeometric(
    const double& x[],        // array with the values of random variable
    const double p,           // parameter of the distribution (probability of event)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the probability function
);
```

Calculates the value of the probability distribution function for geometric law with the *p* parameter for an array of random variables *x[]*. In case of error it returns false.

```
bool MathCumulativeDistributionGeometric(
    const double& x[],        // array with the values of random variable
    const double p,           // parameter of the distribution (probability of event)
    double& result[]          // array for values of the probability function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if tail=true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileGeometric

For the specified *probability*, the function calculates the inverse value of distribution function for geometric law with the *p* parameter. In case of error it returns [NaN](#).

```
double MathQuantileGeometric(
    const double probability, // probability value of random variable occurrence
    const double p,          // parameter of the distribution (probability of event)
    const bool tail,         // flag of calculation, if false, then calculation is for left tail
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation is for log scale
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the inverse value of distribution function for geometric law with the *p* parameter. In case of error it returns [NaN](#).

```
double MathQuantileGeometric(
    const double probability, // probability value of random variable occurrence
    const double p,          // parameter of the distribution (probability of event)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for geometric law with the *p* parameter. In case of error it returns false. Analog of the [qgeom\(\)](#) in R.

```
double MathQuantileGeometric(
    const double& probability[], // array with probability values of random variable
    const double p,             // parameter of the distribution (probability of event)
    const bool tail,            // flag of calculation, if false, then calculation is for left tail
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation is for log scale
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for geometric law with the *p* parameter. In case of error it returns false.

```
bool MathQuantileGeometric(
    const double& probability[], // array with probability values of random variable
    const double p,             // parameter of the distribution (probability of event)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomGeometric

Generates a pseudorandom variable distributed according to the law of geometric distribution with the p parameter. In case of error it returns [NaN](#).

```
double MathRandomGeometric(  
    const double p,           // parameter of the distribution (probability of event occurrence)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of geometric distribution with the p parameter. In case of error it returns false. Analog of the [rgeom\(\)](#) in R.

```
bool MathRandomGeometric(  
    const double p,           // parameter of the distribution (probability of event occurrence)  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

p

[in] Parameter of the distribution (probability of event occurrence in one test).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsGeometric

Calculates the theoretical numerical values of the first 4 moments of the geometric distribution with the p parameter.

```
double MathMomentsGeometric(  
    const double p,           // parameter of the distribution (probability of event occurrence in one test)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

p

[in] Parameter of the distribution (probability of event occurrence in one test).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

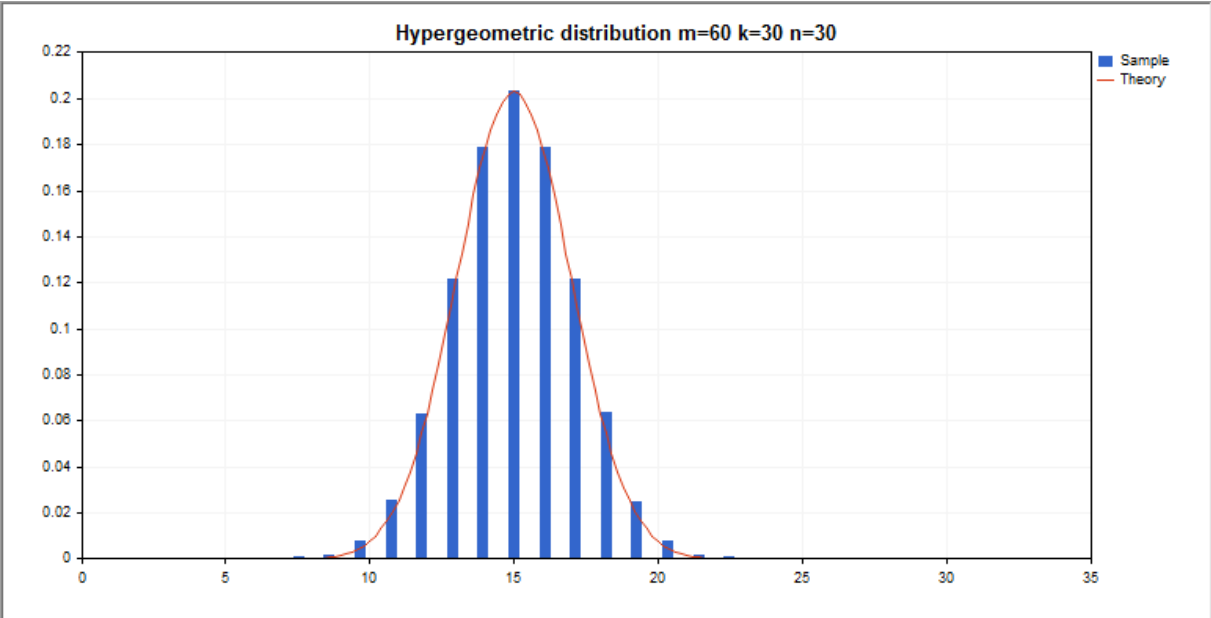
Hypergeometric distribution

This section contains functions for working with hypergeometric distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the hypergeometric law. The hypergeometric distribution is defined by the following formula:

$$f_{Hypergeometric}(x | m, k, n) = \frac{\binom{k}{x} \binom{m-k}{n-x}}{\binom{m}{n}}$$

where:

- x — value of the random variable (integer)
- m — total number of objects
- k — number of objects with the desired characteristic
- n — number of object draws



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityHypergeometric	Calculates the probability density function of the hypergeometric distribution
MathCumulativeDistributionHypergeometric	Calculates the value of the hypergeometric probability distribution function
MathQuantileHypergeometric	Calculates the value of the inverse hypergeometric distribution function for the specified probability

<u>MathRandomHypergeometric</u>	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the hypergeometric law
<u>MathMomentsHypergeometric</u>	Calculates the theoretical numerical values of the first 4 moments of the hypergeometric distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\Hypergeometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double m_par=60;      // the total number of objects
input double k_par=30;      // the number of objects with the desired characteristic
input double n_par=30;      // the number of object draws
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;          // the number of values in the sample
    int ncells=15;          // the number of intervals in the histogram
    double x[];             // centers of the histogram intervals
    double y[];             // the number of values from the sample falling within the int
    double data[];          // sample of random values
    double max,min;         // the maximum and minimum values in the sample
//--- obtain a sample from the hypergeometric distribution
    MathRandomHypergeometric(m_par,k_par,n_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(0,n_par,1,x2);
    MathProbabilityDensityHypergeometric(x2,m_par,k_par,n_par,false,y2);

```



```

//--- set the scale
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- output charts
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Hypergeometric distribution m=%G k=%G n=%G",m,
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```



```

        frequency[ind]++;
    }
    return (true);
}
//+-----+
//|  Calculates values for sequence generation          |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //-- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //-- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //-- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}

```


MathProbabilityDensityHypergeometric

Calculates the value of the probability mass function of hypergeometric distribution with the m , k and n parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityHypergeometric(
    const double x,           // value of random variable (integer)
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of hypergeometric distribution with the m , k and n parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathProbabilityDensityHypergeometric(
    const double x,           // value of random variable (integer)
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of hypergeometric distribution with the m , k and n parameters for an array of random variables $x[]$. In case of error it returns false. Analog of the [dhyper\(\)](#) in R.

```
bool MathProbabilityDensityHypergeometric(
    const double& x[],        // array with the values of random variable
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability mass function of hypergeometric distribution with the m , k and n parameters for an array of random variables $x[]$. In case of error it returns false.

```
bool MathProbabilityDensityHypergeometric(
    const double& x[],        // array with the values of random variable
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionHypergeometric

Calculates the value of the probability distribution function for hypergeometric law with the m , k and n parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathCumulativeDistributionHypergeometric(
    const double x,           // value of random variable (integer)
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for hypergeometric law with the m , k and n parameters for a random variable x . In case of error it returns [NaN](#).

```
double MathCumulativeDistributionHypergeometric(
    const double x,           // value of random variable (integer)
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability distribution function for hypergeometric law with the m , k and n parameters for an array of random variables $x[]$. In case of error it returns false. Analog of the [dhyper\(\)](#) in R.

```
bool MathCumulativeDistributionHypergeometric(
    const double& x[],        // array with the values of random variable
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]         // array for values of the distribution function
);
```

Calculates the value of the probability distribution function for hypergeometric law with the m , k and n parameters for an array of random variables $x[]$. In case of error it returns false.

```
bool MathCumulativeDistributionHypergeometric(
    const double& x[],        // array with the values of random variable
    const double m,           // total number of objects (integer)
    const double k,           // number of objects with the desired characteristic
    const double n,           // number of object draws (integer)
    double& result[]         // array for values of the distribution function
);
```


Parameters*x*

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding *x* is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if *log_mode*=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the distribution function.

MathQuantileHypergeometric

For the specified *probability*, the function calculates the inverse value of distribution function for hypergeometric law with the *m*, *k* and *n* parameters. In case of error it returns [NaN](#).

```
double MathQuantileHypergeometric(
    const double probability, // probability value of random variable occurrence
    const double m,          // total number of objects (integer)
    const double k,          // number of objects with the desired characteristic
    const double n,          // number of object draws (integer)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the inverse value of distribution function for hypergeometric law with the *m*, *k* and *n* parameters. In case of error it returns [NaN](#).

```
double MathQuantileHypergeometric(
    const double probability, // probability value of random variable occurrence
    const double m,          // total number of objects (integer)
    const double k,          // number of objects with the desired characteristic
    const double n,          // number of object draws (integer)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for hypergeometric law with the *m*, *k* and *n* parameters. In case of error it returns false. Analog of the [ghyper\(\)](#) in R.

```
double MathQuantileHypergeometric(
    const double& probability[], // array with probability values of random variable
    const double m,             // total number of objects (integer)
    const double k,             // number of objects with the desired characteristic
    const double n,             // number of object draws (integer)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for hypergeometric law with the *m*, *k* and *n* parameters. In case of error it returns false.

```
bool MathQuantileHypergeometric(
    const double& probability[], // array with probability values of random variable
    const double m,             // total number of objects (integer)
    const double k,             // number of objects with the desired characteristic
    const double n,             // number of object draws (integer)
    double& result[]            // array with values of quantiles
);
```



```
);
```

Parameters*probability*

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

tail

[in] Flag of calculation, if tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomHypergeometric

Generates a pseudorandom variable distributed according to the law of hypergeometric distribution with the *m*, *n* and *k* parameters. In case of error it returns [NaN](#).

```
double MathRandomHypergeometric(  
    const double m,           // total number of objects (integer)  
    const double k,           // number of objects with the desired characteristic  
    const double n,           // number of object draws (integer)  
    int& error_code           // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of hypergeometric distribution with the *m*, *n* and *k* parameters. In case of error it returns false. Analog of the [rgeom\(\)](#) in R.

```
bool MathRandomHypergeometric(  
    const double m,           // total number of objects (integer)  
    const double k,           // number of objects with the desired characteristic  
    const double n,           // number of object draws (integer)  
    const int data_count,     // amount of required data  
    double& result[]          // array with values of pseudorandom variables  
);
```

Parameters

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsHypergeometric

Calculates the theoretical numerical values of the first 4 moments of the hypergeometric distribution with the m, n and k parameters.

```
double MathMomentsHypergeometric(  
    const double m,           // total number of objects (integer)  
    const double k,           // number of objects with the desired characteristic  
    const double n,           // number of object draws (integer)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

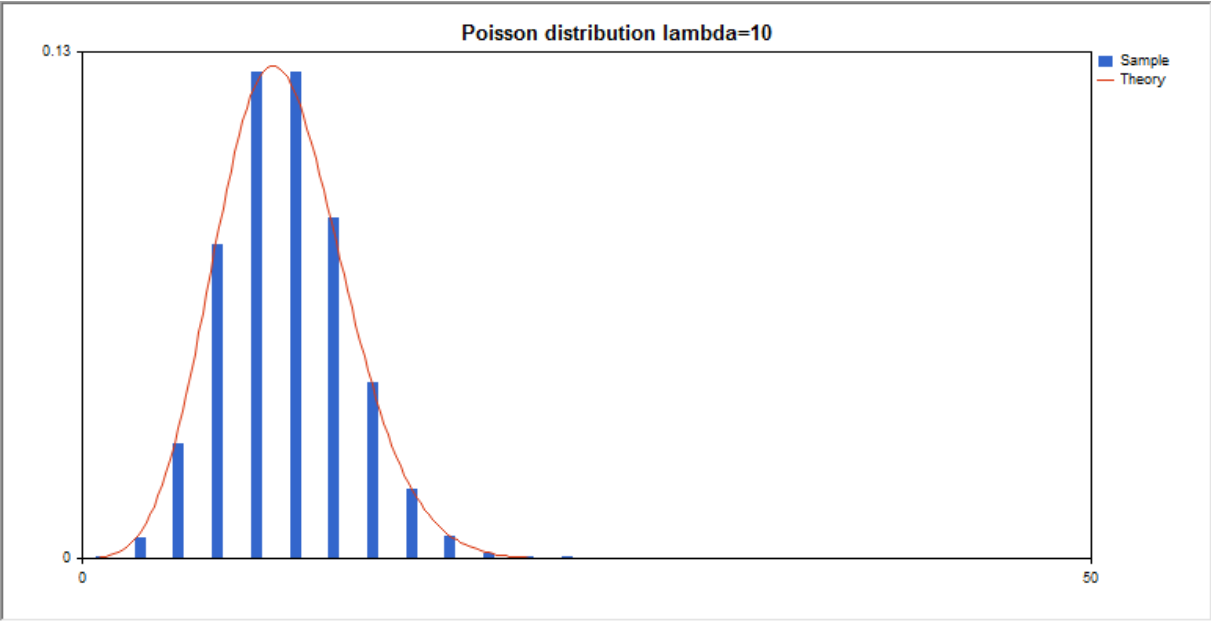
Poisson distribution

This section contains functions for working with Poisson distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Poisson law. The Poisson distribution is defined by the following formula:

$$f_{Poisson}(x|\lambda) = \frac{\lambda^x}{x!} e^{-\lambda}$$

where:

- x – value of the random variable
- λ – parameter of the distribution (mean)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityPoisson	Calculates the probability density function of the Poisson distribution
MathCumulativeDistributionPoisson	Calculates the value of the Poisson probability distribution function
MathQuantilePoisson	Calculates the value of the inverse Poisson distribution function for the specified probability
MathRandomPoisson	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Poisson law
MathMomentsPoisson	Calculates the theoretical numerical values of the first 4 moments of the Poisson distribution

Example:

```

#include <Graphics\Graphic.mqh>
#include <Math\Stat\Poisson.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double lambda_par=10;      // parameter of the distribution (mean)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- hide the price chart
    ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
    MathSrand(GetTickCount());
//--- generate a sample of the random variable
    long chart=0;
    string name="GraphicNormal";
    int n=100000;      // the number of values in the sample
    int ncells=13;     // the number of intervals in the histogram
    double x[];        // centers of the histogram intervals
    double y[];        // the number of values from the sample falling within the int
    double data[];     // sample of random values
    double max,min;    // the maximum and minimum values in the sample
//--- obtain a sample from the Poisson distribution
    MathRandomPoisson(lambda_par,n,data);
//--- calculate the data to plot the histogram
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
    double step;
    GetMaxMinStepValues(max,min,step);
    PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
    double x2[];
    double y2[];
    MathSequence(0,int(MathCeil(max)),1,x2);
    MathProbabilityDensityPoisson(x2,lambda_par,false,y2);
//--- set the scale
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- output charts
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);

```



```

else
    graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("Poisson distribution lambda=%G",lambda_par));
    graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
    graphic.YAxis().AutoScale(false);
    graphic.YAxis().Max(NormalizeDouble(theor_max,2));
    graphic.YAxis().Min(0);
//--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
    graphic.CurvePlotAll();
//--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency[],
                           double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- define the interval centers
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- fill the frequencies of falling within the interval
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |

```



```
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
    //-- calculate the absolute range of the sequence to obtain the precision of normalization
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
    //-- normalize the maximum and minimum values to the specified precision
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
    //-- sequence generation step is also set based on the specified precision
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```


MathProbabilityDensityPoisson

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityPoisson(
    const double x,           // value of random variable (integer)
    const double lambda,      // parameter of the distribution (mean)
    const bool log_mode,      // calculate the logarithm of the value, if log_mode=true
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathProbabilityDensityPoisson(
    const double x,           // value of random variable (integer)
    const double lambda,      // parameter of the distribution (mean)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for an array of random variables x[]. In case of error it returns false. Analog of the [dhyper\(\)](#) in R.

```
bool MathProbabilityDensityPoisson(
    const double& x[],        // array with the values of random variable
    const double lambda,      // parameter of the distribution (mean)
    const bool log_mode,      // flag to calculate the logarithm of the value, if log_mode=true
    double& result[]         // array for values of the probability density function
);
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathProbabilityDensityPoisson(
    const double& x[],        // array with the values of random variable
    const double lambda,      // parameter of the distribution (mean)
    double& result[]         // array for values of the probability density function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

lambda

[in] Parameter of the distribution (mean).

log_mode

[in] Flag to calculate the logarithm of the value. If `log_mode=true`, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionPoisson

Calculates the value of the Poisson distribution function with the lambda parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionPoisson(
    const double x,           // value of random variable (integer)
    const double lambda,      // parameter of the distribution (mean)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Poisson distribution function with the lambda parameter for a random variable x. In case of error it returns [NaN](#).

```
double MathCumulativeDistributionPoisson(
    const double x,           // value of random variable (integer)
    const double lambda,      // parameter of the distribution (mean)
    int& error_code           // variable to store the error code
);
```

Calculates the value of the Poisson distribution function with the lambda parameter for an array of random variables x[]. In case of error it returns false. Analog of the [dhyper\(\)](#) in R.

```
bool MathCumulativeDistributionPoisson(
    const double& x[],        // array with the values of random variable
    const double lambda,      // parameter of the distribution (mean)
    const bool tail,          // flag of calculation, if true, then the probability
    const bool log_mode,      // flag to calculate the logarithm of the value, if 1
    double& result[]          // array for values of the distribution function
);
```

Calculates the value of the Poisson distribution function with the lambda parameter for an array of random variables x[]. In case of error it returns false.

```
bool MathCumulativeDistributionPoisson(
    const double& x[],        // array with the values of random variable
    const double lambda,      // parameter of the distribution (mean)
    double& result[]          // array for values of the distribution function
);
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

lambda

[in] Parameter of the distribution (mean).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the distribution function.

MathQuantilePoisson

For the specified *probability*, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns [NaN](#).

```
double MathQuantilePoisson(
    const double probability, // probability value of random variable occurrence
    const double lambda,     // parameter of the distribution (mean)
    const bool tail,         // flag of calculation, if false, then calculation is
    const bool log_mode,     // flag of calculation, if log_mode=true, calculation
    int& error_code          // variable to store the error code
);
```

For the specified *probability*, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns [NaN](#).

```
double MathQuantilePoisson(
    const double probability, // probability value of random variable occurrence
    const double lambda,     // parameter of the distribution (mean)
    int& error_code          // variable to store the error code
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns false. Analog of the [ghyper\(\)](#) in R.

```
double MathQuantilePoisson(
    const double& probability[], // array with probability values of random variable
    const double lambda,        // parameter of the distribution (mean)
    const bool tail,            // flag of calculation, if false, then calculation is
    const bool log_mode,        // flag of calculation, if log_mode=true, calculation
    double& result[]            // array with values of quantiles
);
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns false.

```
bool MathQuantilePoisson(
    const double& probability[], // array with probability values of random variable
    const double lambda,        // parameter of the distribution (mean)
    double& result[]            // array with values of quantiles
);
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

lambda

[in] Parameter of the distribution (mean).

tail

[in] Flag of calculation, if tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomPoisson

Generates a pseudorandom variable distributed according to the law of Poisson distribution with the lambda parameter. In case of error it returns [NaN](#).

```
double MathRandomPoisson(  
    const double lambda,          // parameter of the distribution (mean)  
    int& error_code               // variable to store the error code  
);
```

Generates pseudorandom variables distributed according to the law of Poisson distribution with the lambda parameter. In case of error it returns false. Analog of the [rgeom\(\)](#) in R.

```
bool MathRandomPoisson(  
    const double lambda,          // parameter of the distribution (mean)  
    const int data_count,         // amount of required data  
    double& result[]             // array with values of pseudorandom variables  
);
```

Parameters

lambda

[in] Parameter of the distribution (mean).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsPoisson

Calculates the theoretical numerical values of the first 4 moments of the Poisson distribution with the lambda parameter.

```
double MathMomentsPoisson(  
    const double lambda,      // parameter of the distribution (mean)  
    double& mean,             // variable for the mean  
    double& variance,         // variable for the variance  
    double& skewness,         // variable for the skewness  
    double& kurtosis,         // variable for the kurtosis  
    int& error_code           // variable for the error code  
);
```

Parameters

lambda

[in] Parameter of the distribution (mean).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

Subfunctions

Group of functions that perform basic mathematical operations: calculation of the gamma function, beta function, factorial, exponential, logarithms with different bases, square root, etc.

They provide the ability to process both individual numeric values (real and integer) and arrays of such values (with output of the results to a separate or the original array).

Function	Description
<u>MathRandomNonZero</u>	Returns a random number with a floating point in the range from 0.0 to 1.0.
<u>MathMoments</u>	Calculates the first 4 moments of array elements: mean, variance, skewness, kurtosis.
<u>MathPowInt</u>	Raises a number to the specified integer power.
<u>MathFactorial</u>	Calculates the factorial of the specified integer.
<u>MathTrunc</u>	Calculates the integer part of the specified number or array elements.
<u>MathRound</u>	Rounds a number or an array of numbers to the specified number of decimal places.
<u>MathArctan2</u>	Calculates the angle, the tangent of which is equal to the ratio of the two specified numbers in the range of $[-\pi, \pi]$.
<u>MathGamma</u>	Calculates the value of the gamma function.
<u>MathGammaLog</u>	Calculates the logarithm of the gamma function.
<u>MathBeta</u>	Calculates the value of the beta function.
<u>MathBetaLog</u>	Calculates the logarithm of the beta function.
<u>MathBetaIncomplete</u>	Calculates the value of the incomplete beta function.
<u>MathGammaIncomplete</u>	Calculates the value of the incomplete gamma function.
<u>MathBinomialCoefficient</u>	Calculates the binomial coefficient.
<u>MathBinomialCoefficientLog</u>	Calculates the logarithm of the binomial coefficient.
<u>MathHypergeometric2F2</u>	Calculates the value of the hypergeometric function.
<u>MathSequence</u>	Generates a sequence based on values: the first element, the last element, the step of the sequence.

<u>MathSequenceByCount</u>	Generates a sequence based on values: the first element, the last element, the number of elements in the sequence.
<u>MathReplicate</u>	Generates a repeating sequence of values.
<u>MathReverse</u>	Generates an array of values with reverse order of elements.
<u>MathIdentical</u>	Compares two arrays of values and returns true if all elements match.
<u>MathUnique</u>	Generates an array with unique values only.
<u>MathQuickSortAscending</u>	Function for sorting in ascending order.
<u>MathQuickSortDescending</u>	Function for sorting in descending order.
<u>MathQuickSort</u>	Function for sorting.
<u>MathOrder</u>	Generates an array with permutation according to order of the array elements after sorting.
<u>MathBitwiseNot</u>	Calculates the result of bitwise NOT operation for array elements.
<u>MathBitwiseAnd</u>	Calculates the result of bitwise AND operation for elements of arrays.
<u>MathBitwiseOr</u>	Calculates the result of bitwise OR operation for elements of arrays.
<u>MathBitwiseXor</u>	Calculates the result of bitwise XOR operation for elements of arrays.
<u>MathBitwiseShiftL</u>	Calculates the result of bitwise SHL operation for array elements.
<u>MathBitwiseShiftR</u>	Calculates the result of bitwise SHR operation for array elements.
<u>MathCumulativeSum</u>	Generates an array with the cumulative sums.
<u>MathCumulativeProduct</u>	Generates an array with the cumulative products.
<u>MathCumulativeMin</u>	Generates an array with the cumulative minima.
<u>MathCumulativeMax</u>	Generates an array with the cumulative maxima.
<u>MathSin</u>	Calculates the values of the sin(x) function for array elements.
<u>MathCos</u>	Calculates the values of the cos(x) function for array elements.

<u>MathTan</u>	Calculates the values of the $\tan(x)$ function for array elements.
<u>MathArcsin</u>	Calculates the values of the $\arcsin(x)$ function for array elements.
<u>MathArccos</u>	Calculates the values of the $\arccos(x)$ function for array elements.
<u>MathArctan</u>	Calculates the values of the $\arctan(x)$ function for array elements.
<u>MathSinPi</u>	Calculates the values of the $\sin(\pi \cdot x)$ function for array elements.
<u>MathCosPi</u>	Calculates the values of the $\cos(\pi \cdot x)$ function for array elements.
<u>MathTanPi</u>	Calculates the values of the $\tan(\pi \cdot x)$ function for array elements.
<u>MathAbs</u>	Calculates the absolute values of array elements.
<u>MathCeil</u>	Returns the nearest larger integers for array elements.
<u>MathFloor</u>	Returns the nearest smaller integers for array elements.
<u>MathSqrt</u>	Calculates the square roots of array elements.
<u>MathExp</u>	Calculates the values of the $\exp(x)$ function for array elements.
<u>MathPow</u>	Calculates the values of the $\text{pow}(x, \text{power})$ function for array elements.
<u>MathLog</u>	Calculates the values of the $\log(x)$ function for array elements.
<u>MathLog2</u>	Calculates the logarithm to the base 2 for the array elements.
<u>MathLog10</u>	Calculates the logarithm to the base 10 for the array elements.
<u>MathDifference</u>	Generates an array with element differences of $y[i] = x[i + \text{lag}] - x[i]$.
<u>MathSample</u>	Generates a random sample from the array elements.
<u>MathTukeySummary</u>	Calculates the Tukey's five-number summary for the array elements.
<u>MathRange</u>	Calculates the minima and maxima of array elements.

<u>MathMin</u>	Returns the minimum value of all array elements.
<u>MathMax</u>	Returns the maximum value of all array elements.
<u>MathSum</u>	Returns the sum of array elements.
<u>MathProduct</u>	Returns the product of array elements.
<u>MathStandardDeviation</u>	Calculates the standard deviation of array elements.
<u>MathAverageDeviation</u>	Calculates the average absolute deviation of array elements.
<u>MathMedian</u>	Calculates the median value of array elements.
<u>MathMean</u>	Calculates the mean values of array elements.
<u>MathVariance</u>	Calculates the variance of the array elements.
<u>MathSkewness</u>	Calculates the skewness of the array elements.
<u>MathKurtosis</u>	Calculates the kurtosis of the array elements.
<u>MathLog1p</u>	Calculates the values of the $\log(1+x)$ function for array elements.
<u>MathExpml</u>	Calculates the values of the $\exp(x)-1$ function for array elements.
<u>MathSinh</u>	Calculates the values of the $\sinh(x)$ function for array elements.
<u>MathCosh</u>	Calculates the values of the $\cosh(x)$ function for array elements.
<u>MathTanh</u>	Calculates the values of the $\tanh(x)$ function for array elements.
<u>MathArcsinh</u>	Calculates the values of the $\operatorname{arsinh}(x)$ function for array elements.
<u>MathArccosh</u>	Calculates the values of the $\operatorname{arcosh}(x)$ function for array elements.
<u>MathArctanh</u>	Calculates the values of the $\operatorname{arctanh}(x)$ function for array elements.
<u>MathSignif</u>	Rounds a value to the specified number of digits in the mantissa.
<u>MathRank</u>	Calculates the ranks of array elements.
<u>MathCorrelationPearson</u>	Calculates the Pearson's correlation coefficient.
<u>MathCorrelationSpearman</u>	Calculates the Spearman's correlation coefficient.

<u>MathCorrelationKendall</u>	Calculates the Kendall's correlation coefficient.
<u>MathQuantile</u>	Calculates sample quantiles corresponding to the specified probabilities.
<u>MathProbabilityDensityEmpirical</u>	Calculates the empirical probability density function for random values.
<u>MathCumulativeDistributionEmpirical</u>	Calculates the empirical cumulative distribution function for random values.

MathRandomNonZero

Returns a random number with a floating point in the range from 0.0 to 1.0.

```
double MathRandomNonZero()
```

Return Value

Random number with a floating point in the range from 0.0 to 1.0.

MathMoments

Calculates the first 4 moments of array elements: mean, variance, skewness, kurtosis.

```
bool MathMoments (
    const double& array[],           // array of values
    double& mean,                   // variable for the mean
    double& variance,               // variable for the variance
    double& skewness,               // variable for the skewness
    double& kurtosis,               // variable for the kurtosis
    const int start=0,               // initial index
    const int count=WHOLE_ARRAY     // the number of elements
)
```

Parameters

array[]

[in] Array of values.

mean

[out] Variable for the mean (1st moment).

variance

[out] Variable for the variance (2nd moment).

skewness

[out] Variable for the skewness (3rd moment).

kurtosis

[out] Variable for the kurtosis (4th moment).

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

MathPowInt

Raises a number to the specified integer power.

```
double MathPowInt(  
    const double x,      // value of the number  
    const int    power   // power to raise to  
)
```

Parameters

x

[in] Double-precision floating-point number to be raised to the power.

power

[in] Integer specifying the power.

Return Value

The number *x*, raised to the specified power.

MathFactorial

Calculates the factorial of the specified integer.

```
double MathFactorial(  
    const int n    // value of the number  
)
```

Parameters

n

[in] Integer number, the factorial of which is to be calculated.

Return Value

The factorial of the number.

MathTrunc

Calculates the integer part of the specified number or array elements.

Version for working with a double-precision floating-point number:

```
double MathTrunc(  
    const double x           // value of the number  
)
```

Return Value

The integer part of the specified number.

Version for working with an array of double-precision floating-point numbers. The results are output to a new array:

```
bool MathTrunc(  
    const double& array[],    // array of values  
    double& result[]         // array of results  
)
```

Return Value

Returns true if successful, otherwise false.

Version for working with an array of double-precision floating-point numbers. The results are output to the original array:

```
bool MathTrunc(  
    double& array[]          // array of values  
)
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Double-precision floating-point number, the integer part of which is to be obtained.

array[]

[in] Array of double-precision floating-point numbers, the integer parts of which are to be obtained.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

MathRound

Rounds a double-precision floating-point number or an array of such numbers to the specified number of decimal places.

Version for rounding a double-precision floating-point number to the specified number of decimal places:

```
double MathRound(  
    const double x,           // value of the number  
    const int digits          // the number of decimal places  
)
```

Return Value

A number nearest to the *x* parameter, with the number of decimal places equal to *digits*.

Version for rounding an array of double-precision floating-point numbers to the specified number of decimal places. The results are output to a new array.

```
bool MathRound(  
    const double& array[],    // array of values  
    int digits,              // the number of decimal places  
    double& result[]         // array of results  
)
```

Return Value

Returns true if successful, otherwise false.

Version for rounding an array of double-precision floating-point numbers to the specified number of decimal places. The results are output to the original array.

```
bool MathRound(  
    double& array[],          // array of values  
    int digits                // the number of decimal places  
)
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Double-precision floating-point number to be rounded.

digits

[in] The number of decimal places in the returned value.

array[]

[in] Array of double-precision floating-point numbers to be rounded.

array[]

[out] Array of output values.

```
result[]
```

[out] Array of output values.

MathArctan2

Returns the arctangent of the quotient of two arguments (x, y).

Version for working with the ratio of the two specified numbers (x, y):

```
double MathArctan2(
    const double    y,          // Y coordinate
    const double    x          // X coordinate
)
```

Return Value

Angle θ , measured in radians, so that $-\pi \leq \theta \leq \pi$ and $\tan(\theta) = y$ or x , where (x, y) is a point in a Cartesian coordinate system.

Version for working with the ratio of the element pairs from the x and y arrays:

```
bool MathArctan2(
    const double&    x[],        // array of x values
    const double&    y[],        // array of y values
    double&          result[]    // array of results
)
```

Return Value

Returns true if successful, otherwise false.

Parameters

y

[in] The Y coordinate of the point.

x

[in] The X coordinate of the point.

x[]

[in] Array of X coordinates of the points.

y[]

[in] Array of Y coordinates of the points.

result[]

[out] Array to output the results

Notes

Please note the following.

- For (x, y) in the quadrant 1, the return value will be: $0 < \theta < \pi/2$.
- For (x, y) in the quadrant 2, the return value will be: $\pi/2 < \theta \leq \pi$.
- For (x, y) in the quadrant 3, the return value will be: $-\pi < \theta < -\pi/2$.
- For (x, y) in the quadrant 4, the return value will be: $-\pi/2 < \theta < 0$.

The return value for the points outside these quadrants is indicated below.

- If y is 0 and x is not negative, then $\theta = 0$.
- If y is 0 and x is negative, then $\theta = \pi$.
- If y is a positive number, and x is 0, then $\theta = \pi/2$.
- If y is negative and x is 0, then $\theta = -\pi/2$.
- If y is 0 and x is 0, then $\theta = -\pi/2$.

If the value of the x or y parameter is NaN, or if the values of the x and y parameters are equal to the value PositiveInfinity or NegativeInfinity, the method returns the NaN value.

MathGamma

Calculates the value of the gamma function for the real argument x .

```
double MathGamma(  
    const double x      // argument of the function  
)
```

Parameters

x

[in] The real argument of the function.

Return Value

The value of the gamma function.

MathGammaLog

Calculates the logarithm of the gamma function for the real argument x.

```
double MathGammaLog(  
    const double x    // argument of the function  
)
```

Parameters

x

[in] The real argument of the function.

Return Value

Logarithm of the function.

MathBeta

Calculates the value of the beta function for the real arguments a and b.

```
double MathBeta(  
    const double a,      // the first argument of the function  
    const double b      // the second argument of the function  
)
```

Parameters

a

[in] The a argument of the function.

b

[in] The b argument of the function.

Return Value

Value of the function.

MathBetaLog

Calculates the logarithm of the beta function for the real arguments a and b.

```
double MathBetaLog(  
    const double a,      // the first argument of the function  
    const double b      // the second argument of the function  
)
```

Parameters

a

[in] The a argument of the function.

b

[in] The b argument of the function.

Return Value

Logarithm of the function.

MathBetaIncomplete

Calculates the value of the incomplete beta function.

```
double MathBetaIncomplete(  
    const double x,      // argument of the function  
    const double p,      // the first parameter of the function  
    const double q       // the second parameter of the function  
)
```

Parameters

x

[in] The argument of the function.

p

[in] The first parameter of the beta function, must be >0.0.

q

[in] The second parameter of the beta function, must be >0.0.

Return Value

Value of the function.

MathGammaIncomplete

Calculates the value of the incomplete gamma function.

```
double MathGammaIncomplete (  
    double x,           // argument of the function  
    double alpha        // parameter of the function  
)
```

Parameters

x

[in] The argument of the function.

alpha

[in] The parameter of the incomplete gamma function.

Return Value

Value of the function.

MathBinomialCoefficient

Calculates the binomial coefficient: $C(n,k)=n!/(k!(n-k)!)$.

```
long MathBinomialCoefficient(  
    const int n,          // the total number of elements  
    const int k           // the number of elements in combination  
)
```

Parameters

n

[in] The number of elements.

k

[in] The number of elements for each combination.

Return Value

The number of combinations of N choose K.

MathBinomialCoefficientLog

Calculates the logarithm of the binomial coefficient: $\text{Log}(C(n,k)) = \text{Log}(n! / (k! * (n-k)!))$

Version for integer arguments:

```
double MathBinomialCoefficientLog(  
    const int    n,          // the total number of elements  
    const int    k          // the number of elements in combination  
)
```

Version for real arguments:

```
double MathBinomialCoefficientLog(  
    const double n,          // the total number of elements  
    const double k          // the number of elements in combination  
)
```

Parameters

n

[in] The number of elements.

k

[in] The number of elements for each combination.

Return Value

The logarithm of $C(n,k)$.

MathHypergeometric2F2

Calculates the value of the Hypergeometric_2F2 (a, b, c, d, z) function using the Taylor's method.

```
double MathHypergeometric2F2(  
    const double a,      // the first parameter of the function  
    const double b,      // the second parameter of the function  
    const double c,      // the third parameter of the function  
    const double d,      // the fourth parameter of the function  
    const double z       // the fifth parameter of the function  
)
```

Parameters

a

[in] The first parameter of the function.

b

[in] The second parameter of the function.

c

[in] The third parameter of the function.

d

[in] The fourth parameter of the function.

z

[in] The fifth parameter of the function.

Return Value

Value of the function.

MathSequence

Generates a sequence of values based on the following values: the first element, the last element, the step of the sequence.

Version for working with real values:

```
bool MathSequence(  
    const double from,      // initial value  
    const double to,        // final value  
    const double step,      // step  
    double& result[]        // array of results  
)
```

Version for working with integer values:

```
bool MathSequence(  
    const int from,         // initial value  
    const int to,           // final value  
    const int step,         // step  
    int& result[]           // array of results  
)
```

Parameters

from

[in] The first value of the sequence

to

[in] The last value of the sequence

step

[in] The step of the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathSequenceByCount

Generates a sequence of values based on the following values: the first element, the last element, the number of elements in the sequence.

Version for working with real values:

```
bool MathSequenceByCount (
    const double from,      // initial value
    const double to,        // final value
    const int count,        // count
    double& result[]        // array of results
)
```

Version for working with integer values:

```
bool MathSequenceByCount (
    const int from,         // initial value
    const int to,           // final value
    const int count,        // count
    int& result[]           // array of results
)
```

Parameters

from

[in] The first value of the sequence.

to

[in] The last value of the sequence.

count

[in] The number of elements in the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathReplicate

Generates a repeating sequence of values.

Version for working with real values:

```
bool MathReplicate(  
    const double& array[],    // array of values  
    const int     count,      // number of repetitions  
    double&       result[]    // array of results  
)
```

Version for working with integer values:

```
bool MathReplicate(  
    const int&    array[],    // array of values  
    const int     count,      // number of repetitions  
    int&          result[]    // array of results  
)
```

Parameters

array[]

[in] Array for generating a sequence.

count

[in] The number of the array repetitions in the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathReverse

Generates an array of values with reverse order of elements.

Version for working with real values and with output of the results to a new array:

```
bool MathReverse (
    const double& array[],      // array of values
    double& result[]           // array of results
)
```

Version for working with integer values and with output of the results to a new array:

```
bool MathReverse (
    const int& array[],         // array of values
    int& result[]              // array of results
)
```

Version for working with real values and with output of the results to the original array.

```
bool MathReverse (
    double& array[]            // array of values
)
```

Version for working with integer values and with output of the results to the original array:

```
bool MathReverse (
    int& array[]               // array of values
)
```

Parameters

array[]
[in] Array of values.

array[]
[out] Output array with the reverse order of values.

result[]
[out] Output array with the reverse order of values.

Return Value

Returns true if successful, otherwise false.

MathIdentical

Compares two arrays of values and returns true if all elements match.

Version for working with arrays of real values:

```
bool MathIdentical(  
    const double& array1[],    // the first array of values  
    const double& array2[]    // the second array of values  
)
```

Version for working with arrays of integer values:

```
bool MathIdentical(  
    const int& array1[],      // the first array of values  
    const int& array2[]      // the second array of values  
)
```

Parameters

array1[]

[in] The first array to compare.

array2[]

[in] The second array to compare.

Return Value

Returns true if the arrays are equal, otherwise false.

MathUnique

Generates an array with unique values only.

Version for working with real values:

```
bool MathUnique(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version for working with integer values:

```
bool MathUnique(  
    const int& array[], // array of values  
    int& result[] // array of results  
)
```

Parameters

array[]

[in] The source array.

result[]

[out] Array to output the unique values.

Return Value

Returns true if successful, otherwise false.

MathQuickSortAscending

The function for the simultaneous ascending sorting of the `array[]` and `indices[]` arrays using the QuickSort algorithm.

```
void MathQuickSortAscending(  
    double& array[],    // array of values  
    int& indices[],    // array of indexes  
    int first,          // initial value  
    int last            // final value  
)
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

MathQuickSortDescending

The function for the simultaneous descending sorting of the `array[]` and `indices[]` arrays using the QuickSort algorithm.

```
void MathQuickSortDescending(  
    double& array[],    // array of values  
    int& indices[],    // array of indexes  
    int first,          // initial value  
    int last            // final value  
)
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

MathQuickSort

The function for the simultaneous sorting of the `array[]` and `indices[]` arrays using the QuickSort algorithm.

```
void MathQuickSort(  
    double& array[], // array of values  
    int& indices[], // array of indexes  
    int first, // initial value  
    int last, // final value  
    int mode // direction  
)
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

mode

[in] Direction of sorting (>0 ascending; otherwise, descending).

MathOrder

Generates an integer array with permutation according to order of the array elements after sorting.

Version for working with an array of real values:

```
bool MathOrder(  
    const double& array[], // array of values  
    int& result[] // array of results  
)
```

Version for working with an array of integer values:

```
bool MathOrder(  
    const int& array[], // array of values  
    int& result[] // array of results  
)
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array to output the sorted indexes.

Return Value

Returns true if successful, otherwise false.

MathBitwiseNot

Calculates the result of bitwise NOT operation for array elements.

Version with output of the results to a new array:

```
bool MathBitwiseNot(  
    const int& array[], // array of values  
    int& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathBitwiseNot(  
    int& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

array[]
[out] Array of output values.

result[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathBitwiseAnd

Calculates the result of bitwise AND operation for specified arrays.

```
bool MathBitwiseAnd(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    int& result[] // array of results  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseOr

Calculates the result of bitwise OR operation for specified arrays.

```
bool MathBitwiseOr(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    int& result[] // array of results  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseXor

Calculates the result of bitwise XOR operation for specified arrays.

```
bool MathBitwiseXor(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    int& result[] // array of results  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseShiftL

Calculates the result of bitwise SHL (bitwise shift left) operation for array elements.

Version with output of the results to a new array:

```
bool MathBitwiseShiftL(  
    const int& array[],    // array of values  
    const int  n,          // shift value  
    int&       result[]    // array of results  
)
```

Version with output of the results to the original array:

```
bool MathBitwiseShiftL(  
    int&       array[],    // array of values  
    const int  n           // shift value  
)
```

Parameters

array[]

[in] Array of values.

n

[in] The number of bits to shift.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathBitwiseShiftR

Calculates the result of bitwise SHR (bitwise shift right) operation for array elements.

Version with output of the results to a new array:

```
bool MathBitwiseShiftR(  
    const int& array[],    // array of values  
    const int  n,          // shift value  
    int&       result[]    // array of results  
)
```

Version with output of the results to the original array:

```
bool MathBitwiseShiftR(  
    int&       array[],    // array of values  
    const int  n           // shift value  
)
```

Parameters

array[]

[in] Array of values.

n

[in] The number of bits to shift.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeSum

Generates an array with the cumulative sums.

Version with output of the results to a new array:

```
bool MathCumulativeSum(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCumulativeSum(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

array[]
[out] Array of output values.

result[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeProduct

Generates an array with the cumulative products.

Version with output of the results to a new array:

```
bool MathCumulativeProduct(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCumulativeProduct(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeMin

Generates an array with the cumulative minima.

Version with output of the results to a new array:

```
bool MathCumulativeMin(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCumulativeMin(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeMax

Generates an array with the cumulative maxima.

Version with output of the results to a new array:

```
bool MathCumulativeMax(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCumulativeMax(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSin

Calculates the values of the $\sin(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathSin(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathSin(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCos

Calculates the values of the $\cos(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathCos (
    const double& array[], // array of values
    double& result[] // array of results
)
```

Version with output of the results to the original array:

```
bool MathCos (
    double& array[] // array of values
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTan

Calculates the values of the $\tan(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathTan(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathTan(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArcsin

Calculates the values of the arcsin(x) function for array elements.

Version with output of the results to a new array:

```
bool MathArcsin(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathArcsin(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArccos

Calculates the values of the $\arccos(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathArccos (
    const double&   array[],    // array of values
    double&         result[]    // array of results
)
```

Version with output of the results to the original array:

```
bool MathArccos (
    double&         array[]     // array of values
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArctan

Calculates the values of the $\arctan(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathArctan(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathArctan(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSinPi

Calculates the values of the $\sin(\pi \cdot x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathSinPi(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathSinPi(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCosPi

Calculates the values of the $\cos(\pi \cdot x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathCosPi(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCosPi(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTanPi

Calculates the values of the $\tan(\pi \cdot x)$ function for array elements.

Version with output of the result to a new array:

```
bool MathTanPi(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the result to the original array:

```
bool MathTanPi(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathAbs

Calculates the absolute values of array elements.

Version with output of the results to a new array:

```
bool MathAbs (
    const double& array[], // array of values
    double& result[] // array of results
)
```

Version with output of the results to the original array:

```
bool MathAbs (
    double& array[] // array of values
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCeil

Returns the nearest larger integers for array elements.

Version with output of the results to a new array:

```
bool MathCeil(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCeil(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathFloor

Returns the nearest smaller integers for array elements.

Version with output of the results to a new array:

```
bool MathFloor(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathFloor(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSqrt

Calculates the square roots of array elements.

Version with output of the results to a new array:

```
bool MathSqrt(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathSqrt(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathExp

Calculates the values of the $\exp(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathExp(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathExp(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathPow

Calculates the values of the pow(x, power) function for array elements.

Version with output of the results to a new array:

```
bool MathPow(  
    const double& array[],    // array of values  
    const double power,      // power  
    double& result[]         // array of results  
)
```

Version with output of the results to the original array:

```
bool MathPow(  
    double& array[],          // array of values  
    const double power        // power  
)
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog

Calculates the values of the $\log(x)$ function for array elements.

Version for calculating the natural logarithm with output of the results to a new array.

```
bool MathLog(  
    const double& array[],      // array of values  
    double& result[]           // array of results  
)
```

Version for calculating the natural logarithm with output of the results to the original array.

```
bool MathLog(  
    double& array[]            // array of values  
)
```

Version for calculating the logarithm to a specified base with output of the results to a new array.

```
bool MathLog(  
    const double& array[],      // array of values  
    const double base,          // base of the logarithm  
    double& result[]           // array of results  
)
```

Version for calculating the logarithm to a specified base with output of the results to the original array.

```
bool MathLog(  
    double& array[],            // array of values  
    const double base           // base of the logarithm  
)
```

Parameters

array[]

[in] Array of values.

base

[in] The base of the logarithm.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog2

Calculates the logarithm to the base 2 for the array elements.

Version with output of the results to a new array:

```
bool MathLog2(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathLog2(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog10

Calculates the logarithm to the base 10 for the array elements.

Version with output of the results to a new array:

```
bool MathLog10(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathLog10(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog1p

Calculates the values of the $\log(1+x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathLog1p(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathLog1p(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathDifference

Generates an array with element differences of $y[i]=x[i+lag]-x[i]$.

Version for a single generation of an array of real values:

```
bool MathDifference(
    const double &array[], // array of values
    const int    lag,      // lag
    double       &result[] // array of results
)
```

Version for a single generation of an array of integer values:

```
bool MathDifference(
    const int    &array[], // array of values
    const int    lag,      // lag
    int          &result[] // array of results
)
```

Version for iterated generation of an array of real values (the number of iterations is set in the input parameters):

```
bool MathDifference(
    const double &array[], // array of values
    const int    lag,      // lag
    const int    differences, // number of iterations
    double       &result[] // array of results
)
```

Version for iterated generation of an array of integer values (the number of iterations is set in the input parameters):

```
bool MathDifference(
    const int&    array[], // array of values
    const int    lag,      // lag
    const int    differences, // number of iterations
    int&         result[] // array of results
)
```

Parameters

array[]
[in] Array of values.

lag
[in] Lag parameter.

differences
[in] The number of iterations.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathSample

Generates a random sample from the array elements.

Version for working with an array of real values:

```
bool MathSample(  
    const double& array[],           // array of values  
    const int     count,             // count  
    double&       result[]          // array of results  
)
```

Version for working with an array of integer values:

```
bool MathSample(  
    const int&    array[],           // array of values  
    const int     count,             // count  
    int&          result[]          // array of results  
)
```

Version for working with an array of real values. It is possible to obtain a sample with replacement:

```
bool MathSample(  
    const double& array[],           // array of values  
    const int     count,             // count  
    const bool    replace,           // flag  
    double&       result[]          // array of results  
)
```

Version for working with an array of integer values. It is possible to obtain a sample with replacement:

```
bool MathSample(  
    const int&    array[],           // array of values  
    const int     count,             // count  
    const bool    replace,           // flag  
    int&          result[]          // array of results  
)
```

Version for working with an array of real values, for which the probabilities of sampling are defined.

```
bool MathSample(  
    const double& array[],           // array of values  
    double&       probabilities[],  // array of probabilities  
    const int     count,             // count  
    double&       result[]          // array of results  
)
```

Version for working with an array of integer values, for which the probabilities of sampling are defined.


```

bool MathSample(
    const int&    array[],           // array of values
    double&       probabilities[],   // array of probabilities
    const int     count,             // count
    int&          result[]           // array of results
)

```

Version for working with an array of real values, for which the probabilities of sampling are defined. It is possible to obtain a sample with replacement:

```

bool MathSample(
    const double& array[],           // array of values
    double&       probabilities[],   // array of probabilities
    const int     count,             // count
    const bool    replace,           // flag
    double&       result[]           // array of results
)

```

Version for working with an array of integer values, for which the probabilities of sampling are defined. It is possible to obtain a sample with replacement:

```

bool MathSample(
    const int&    array[],           // array of values
    double&       probabilities[],   // array of probabilities
    const int     count,             // count
    const bool    replace,           // flag
    int&          result[]           // array of results
)

```

Parameters

array[]

[in] Array of integer values.

probabilities[]

[in] Array of probabilities for sampling the elements.

count

[in] The number of elements.

replace

[in] Parameter that allows sampling with replacement.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

Note

The `replace=true` argument allows performing random sampling of the elements with replacement back to the original sequence.

MathTukeySummary

Calculates the Tukey's five-number summary (minimum, lower quartile, median, upper quartile, maximum) for the array elements.

```
bool MathTukeySummary(  
    const double& array[],           // array of values  
    const bool    removeNAN,        // flag  
    double&       minimum,          // minimum value  
    double&       lower_hinge,      // lower quartile  
    double&       median,           // median value  
    double&       upper_hinge,      // upper quartile  
    double&       maximum           // maximum value  
)
```

Parameters

array[]

[in] Array of real values.

removeNAN

[in] Flag that indicates if non-numeric values are to be removed.

minimum

[out] Variable to store the minimum value.

lower_hinge

[out] Variable to store the lower quartile.

median

[out] Variable to store the median value.

upper_hinge

[out] Variable to store the upper quartile.

maximum

[out] Variable to store the maximum value.

Return Value

Returns true if successful, otherwise false.

MathRange

Calculates the minima and maxima of array elements.

```
bool MathRange(  
    const double& array[], // array of values  
    double& min,           // minimum value  
    double& max            // maximum value  
)
```

Parameters

array[]
[in] Array of values.

min
[out] Variable to store the minimum value.

max
[out] Variable to store the maximum value.

Return Value

Returns true if successful, otherwise false.

MathMin

Returns the minimum value of all array elements.

```
double MathMin(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The minimum value.

MathMax

Returns the maximum value of all array elements.

```
double MathMax(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The maximum value.

MathSum

Returns the sum of array elements.

```
double MathSum(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The sum of the elements.

MathProduct

Returns the product of array elements.

```
double MathProduct(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The product of the elements.

MathStandardDeviation

Calculates the standard deviation of array elements.

```
double MathStandardDeviation(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

Standard deviation.

MathAverageDeviation

The function calculates the average absolute deviation of array elements.

```
double MathAverageDeviation(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The average absolute deviation of array elements.

MathMedian

Calculates the median value of array elements.

```
double MathMedian(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The median value.

MathMean

Calculates the mean values of array elements.

```
double MathMean(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

The mean value.

MathVariance

The function calculates the variance (second moment) of array elements.

```
double MathVariance(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

Value of the variance.

MathSkewness

The function calculates the skewness (third moment) of array elements.

```
double MathSkewness(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

Skewness.

MathKurtosis

The function calculates the kurtosis (fourth moment) of array elements.

```
double MathKurtosis(  
    const double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

Return Value

Kurtosis.

MathExpm1

Calculates the values of the $\exp(x)-1$ function for array elements.

Version with output of the results to a new array:

```
bool MathExpm1 (
    const double& array[], // array of values
    double& result[] // array of results
)
```

Version with output of the results to the original array:

```
bool MathExpm1 (
    double& array[] // array of values
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSinh

Calculates the values of the $\sinh(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathSinh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathSinh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCosh

Calculates the values of the cosh(x) function for array elements.

Version with output of the results to a new array:

```
bool MathCosh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathCosh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTanh

Calculates the values of the $\tanh(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathTanh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathTanh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArcsinh

Calculates the values of the $\operatorname{arsinh}(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathArcsinh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathArcsinh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArccosh

Calculates the values of the arccosh(x) function for array elements.

Version with output of the results to a new array:

```
bool MathArccosh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathArccosh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArctanh

Calculates the values of the $\operatorname{arctanh}(x)$ function for array elements.

Version with output of the results to a new array:

```
bool MathArctanh(  
    const double& array[], // array of values  
    double& result[] // array of results  
)
```

Version with output of the results to the original array:

```
bool MathArctanh(  
    double& array[] // array of values  
)
```

Parameters

array[]
[in] Array of values.

result[]
[out] Array of output values.

array[]
[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSignif

Rounds a value to the specified number of digits in the mantissa.

Version for working with a real value:

```
double MathSignif(  
    const double x,           // value  
    const int    digits       // number of decimal places  
)
```

Return Value

The rounded value.

Version for working with an array of real values and with output of the results to a separate array:

```
bool MathSignif(  
    const double& array[],    // array of values  
    int          digits,      // number of decimal places  
    double       result[]     // array of results  
)
```

Return Value

Returns true if successful, otherwise false.

Version for working with an array of real values and with output of the results to the original array:

```
bool MathSignif(  
    double&       array[],    // array of values  
    int          digits       // number of decimal places  
)
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Real value to be rounded.

digits

[in] Number of decimal places.

array[]

[in] Array of real values.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

MathRank

Calculates the ranks of array elements.

Version for working with an array of real values:

```
bool MathRank(  
    const double& array[], // array of values  
    double& rank[] // array of ranks  
)
```

Version for working with an array of integer values:

```
bool MathRank(  
    const int& array[], // array of values  
    double& rank[] // array of ranks  
)
```

Parameters

array[]

[in] Array of values.

rank[]

[out] Array to output the ranks.

Return Value

Returns true if successful, otherwise false.

MathCorrelationPearson

Calculates the Pearson's correlation coefficient.

Version for working with arrays of real values:

```
bool MathCorrelationPearson(  
    const double& array1[], // the first array of values  
    const double& array2[], // the second array of values  
    double& r               // correlation coefficient  
)
```

Version for working with arrays of integer values:

```
bool MathCorrelationPearson(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    double& r            // correlation coefficient  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

r

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathCorrelationSpearman

Calculates the Spearman's correlation coefficient.

Version for working with arrays of real values:

```
bool MathCorrelationSpearman(  
    const double& array1[], // the first array of values  
    const double& array2[], // the second array of values  
    double& r               // correlation coefficient  
)
```

Version for working with arrays of integer values:

```
bool MathCorrelationSpearman(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    double& r           // correlation coefficient  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

r

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathCorrelationKendall

Calculates the Kendall's correlation coefficient.

Version for working with arrays of real values:

```
bool MathCorrelationKendall(  
    const double& array1[], // the first array of values  
    const double& array2[], // the second array of values  
    double& tau           // correlation coefficient  
)
```

Version for working with arrays of integer values:

```
bool MathCorrelationKendall(  
    const int& array1[], // the first array of values  
    const int& array2[], // the second array of values  
    double& tau          // correlation coefficient  
)
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

tau

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathQuantile

Calculates sample quantiles corresponding to the specified probabilities: $Q[i](p) = (1 - \text{gamma}) * x[j] + \text{gamma} * x[j+1]$

```
bool MathQuantile(  
    const double& array[],      // array of values  
    const double& probs[],      // array of probabilities  
    double& quantile[]          // array to output the quantiles  
)
```

Parameters

array[]

[in] Array of values.

probs[]

[in] Array of probabilities.

quantile[]

[out] Array to output the quantiles.

Return Value

Returns true if successful, otherwise false.

MathProbabilityDensityEmpirical

The function calculates the empirical probability density function (pdf) for random values from an array.

```
bool MathProbabilityDensityEmpirical(  
    const double& array[], // array of random values  
    const int count, // the number of pairs  
    double& x[], // array of x values  
    double& pdf[] // array of pdf values  
)
```

Parameters

array[]

[in] Array of random values.

count

[in] The number of (x, pdf(x)) pairs.

x[]

[out] Array to output the x values.

pdf[]

[out] Array to output the pdf(x) values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeDistributionEmpirical

The function calculates the empirical cumulative distribution function (cdf) for random values from an array.

```
bool MathCumulativeDistributionEmpirical(  
    const double& array[], // array of random values  
    const int count, // the number of pairs  
    double& x[], // array of x values  
    double& cdf[] // array of cdf values  
)
```

Parameters

array[]

[in] Array of random values.

count

[in] The number of (x, cdf(x)) pairs.

x[]

[out] Array to output the x values.

cdf[]

[out] Array to output the cdf(x) values.

Return Value

Returns true if successful, otherwise false.

Fuzzy is a library for working with fuzzy logic

Fuzzy logic is a synthesis of the traditional Aristotelian logic when truth is marked as a linguistic variable. Fuzzy logic, equivalent to classical logic, has its own fuzzy logic operations on fuzzy sets defined. There are the same operations for fuzzy sets as well as for ordinary sets, only their calculation is by far more difficult. We should also note that the composition of fuzzy sets constitutes as a fuzzy set.

The main principles of fuzzy logic, setting it apart from classical logic, are the maximum proximity to the reflection of reality and a high level of subjectivity, which can lead to significant errors in calculations.

Fuzzy model (or system) is a mathematical model whose calculation is based on fuzzy logic. Construction of such models is applicable when the subject of study has a weak formalization and its exact mathematical description is too complex or unknown. The quality of these models' output values (error model) directly depends only on the Expert Advisor, which set up this model. The best option to minimize errors is to draw the most complete and comprehensive model and subsequently adjust it with machine learning on a large training set.

The progress of a model construction can be divided into three main stages:

1. Definition of input and output characteristics of a model.
2. Building a knowledge base.
3. Selecting one of the methods of fuzzy inference ([Mamdani](#) or [Sugeno](#)).

The first stage directly effects the consequent two and determines the future operation of the model. A knowledge base or, as sometimes called, [rule](#) base is a set of fuzzy rules type: "if, then" that define the relationship between inputs and outputs of the examined object. The number of rules in the system is not limited and is also determined by the Expert Advisor. The generalized format of fuzzy rules is as follows:

If rule condition, then rule conclusion.

Rule condition describes the current state of the object, and rule conclusion — how this condition affects the object. General view of conditions and conclusions cannot be selected because they are determined by a fuzzy inference.

Each rule in the system has its weight — this characteristic defines the importance of a rule in the model. Weighting factors are assigned to a rule within range [0, 1]. In many examples with fuzzy models, which can be found in the relevant literature, weight data is not specified, but it does not mean that it is not present. In fact, in such case for each rule from the database, the weight is fixed and equals unity. There can be two types of terms and conclusions for each rule:

1. simple — includes one [fuzzy variable](#);
2. complex — includes several fuzzy variables.

Depending on the created knowledge base, the system of fuzzy inference is determined for a model. Fuzzy logical inference is a receipt of conclusion in form of a fuzzy set corresponding to the current value of the inputs with use of knowledge base and fuzzy operations. The two main types of fuzzy inference are Mamdani and Sugeno.

Membership functions

A **membership function** is a function that allows to calculate the membership degree of a random element of the universal set to a fuzzy set. Consequently, the domain of a membership function should be within the range [0, 1].

In most cases, the membership function is continuous and monotonic.

Classes of membership functions	Description
<u>CConstantMembershipFunction</u>	Class for implementing a membership function as a straight line in parallel with the coordinate axis
<u>CCompositeMembershipFunction</u>	Class for implementing a composition of membership functions
<u>CDifferencTwoSigmoidalMembershipFunction</u>	Class for implementing the membership function in the form of a difference between two sigmoid functions with the A1, A2, C1 and C2 parameters
<u>CGeneralizedBellShapedMembershipFunction</u>	Class for implementing a generalized bell-shaped membership function with A, B and C parameters
<u>CNormalCombinationMembershipFunction</u>	Class for implementing a two-sided Gaussian membership function with the B1, B2, Sigma1 and Sigma2 parameters
<u>CNormalMembershipFunction</u>	Class for implementing a symmetrical Gaussian membership function with the B and Sigma parameters
<u>CP_ShapedMembershipFunction</u>	Class for implementing a pi-shaped membership function with the A, B, C and D parameters
<u>CProductTwoSigmoidalMembershipFunction</u>	Class for implementing the membership function in the form of a product of two sigmoid functions with the A1, A2, C1 and C2 parameters
<u>CS_ShapedMembershipFunction</u>	Class for implementing an S-like membership function with the A and B parameters
<u>CTrapezoidMembershipFunction</u>	Class for implementing a trapezoidal membership function with the X1, X2, X3 and X4 parameters
<u>CTriangularMembershipFunction</u>	Class for implementing a triangle membership function with the X1, X2 and X3 parameters
<u>CSigmoidalMembershipFunction</u>	Class for implementing a sigmoid membership function with the A and C parameters

<u>CZ_ShapedMembershipFunction</u>	Class for implementing a z-like membership function with the A and B parameters.
<u>IMembershipFunction</u>	Basic class for all membership function classes.

CConstantMembershipFunction

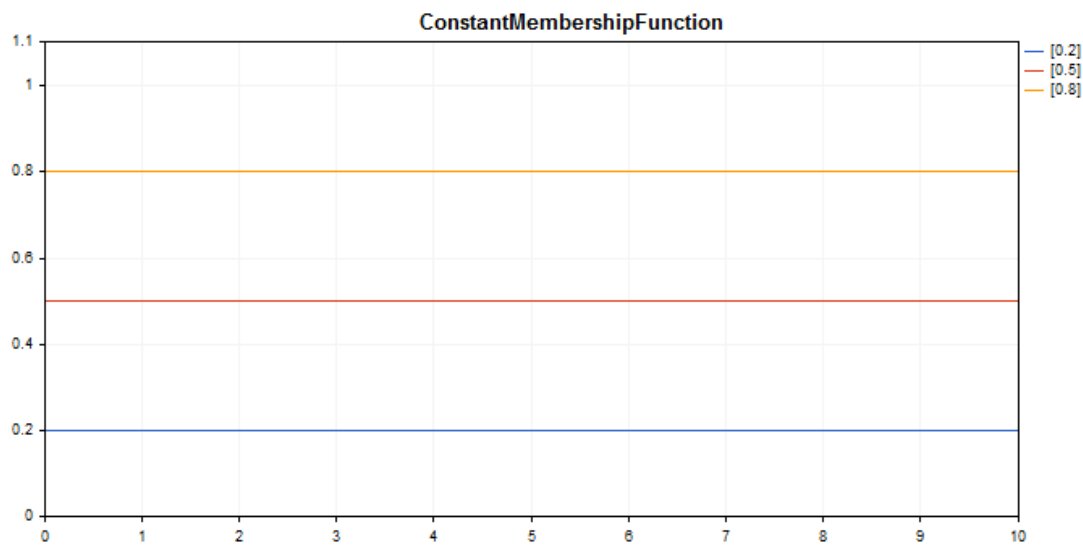
Class for implementing a membership function as a straight line in parallel with the coordinate axis.

Description

The function is described by the equation:

$y(x)=c$

Therefore, the membership degree for the function is the same along the entire numerical axis and is equal to the parameter specified in the constructor.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CConstantMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
- [IMembershipFunction](#)
- CConstantMembershipFunction

Class methods

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     ConstantMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CConstantMembershipFunction func1(0.2);
CConstantMembershipFunction func2(0.5);
CConstantMembershipFunction func3(0.8);
//--- Create wrappers for membership functions
double ConstantMembershipFunction1(double x) { return(func1.GetValue(x)); }
double ConstantMembershipFunction2(double x) { return(func2.GetValue(x)); }
double ConstantMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"ConstantMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"ConstantMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("ConstantMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(ConstantMembershipFunction1,0.0,10.0,1.0,CURVE_LINES,"[0.2]");
graphic.CurveAdd(ConstantMembershipFunction2,0.0,10.0,1.0,CURVE_LINES,"[0.5]");
graphic.CurveAdd(ConstantMembershipFunction3,0.0,10.0,1.0,CURVE_LINES,"[0.8]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
```



```
graphic.CurvePlotAll();  
graphic.Update();  
}
```

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const double x    // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

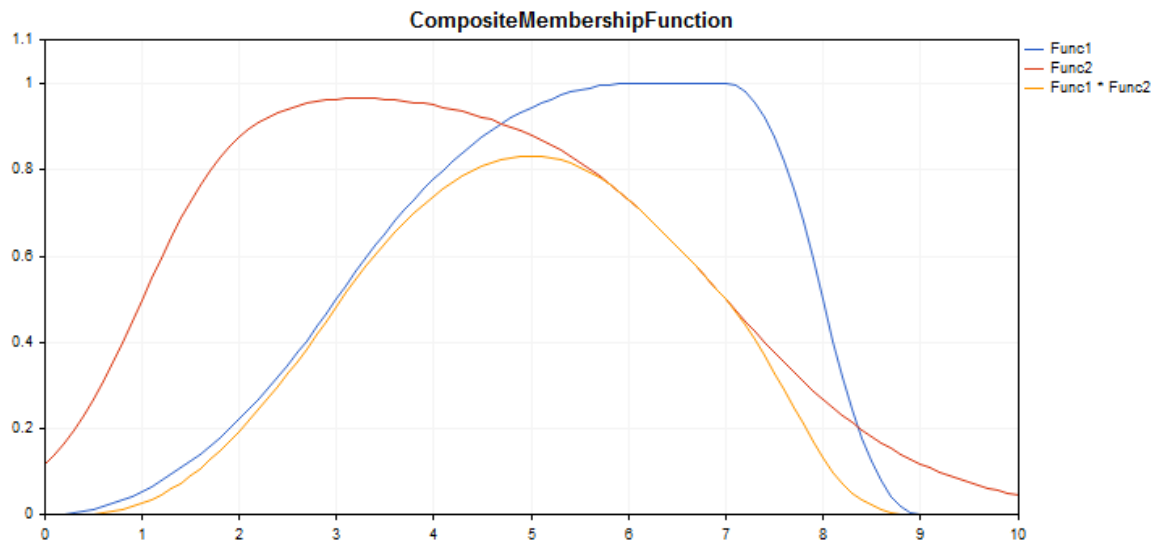
Membership function value.

CCompositeMembershipFunction

Class for implementing a composition of membership functions.

Description

Composition of membership functions is a combination of two or more membership functions using a specified operator.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CCompositeMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
- [IMembershipFunction](#)
- CCompositeMembershipFunction

Class methods

Class method	Description
CompositionType	Sets the composition operator.
MembershipFunctions	Gets the list of membership functions.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     CompositeMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CP_ShapedMembershipFunction func2(0,6,7,9);
CCompositeMembershipFunction composite(ProdMF,GetPointer(func1),GetPointer(func2));
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction(double x) { return(func2.GetValue(x)); }
double CompositeMembershipFunction(double x) { return(composite.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"CompositeMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"CompositeMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("CompositeMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(P_ShapedMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1");
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions,0.0,10.0,0.1,CURVE_LINES,"Func2");
graphic.CurveAdd(CompositeMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1 * Func2");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
```



```
graphic.CurvePlotAll();  
graphic.Update();  
}
```

CompositionType

Sets the composition operator.

```
void CompositionType(  
    MfCompositionType value    // operator type  
)
```

Parameters

value

[in] Composition operator type.

Note

The following operator types are available:

- MinMF (minimum of functions)
- MaxMF (maximum of functions)
- ProdMF (product of functions)
- SumMF (sum of functions)

MembershipFunctions

Gets the list of membership functions included into a composition.

```
CList* MembershipFunctions(  
    void    // list of membership functions  
)
```

Return Value

List of membership functions.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const x    // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

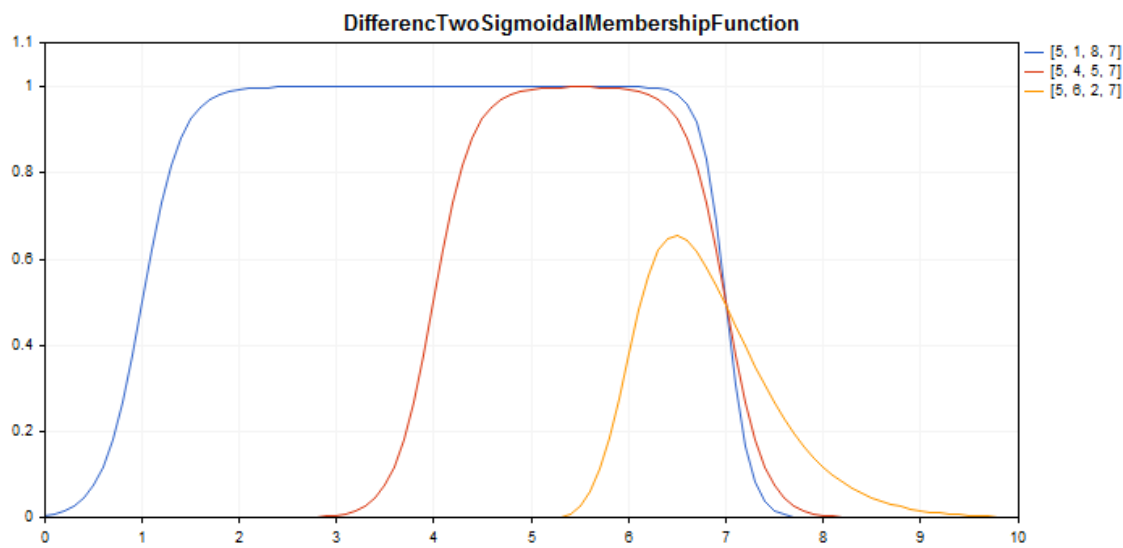
Membership function value.

CDifferencTwoSigmoidalMembershipFunction

Class for implementing the membership function in the form of a difference between two sigmoid functions with the A1, A2, C1 and C2 parameters.

Description

The function is based on a sigmoid curve. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CDifferencTwoSigmoidalMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

[CObject](#)

[IMembershipFunction](#)

CDifferencTwoSigmoidalMembershipFunction

Class methods

Class method	Description
A1	Gets and sets the first membership function slope ratio.

A2	Gets and sets the second membership function slope ratio.
C1	Gets and sets the first membership function inflection coordinate parameter.
C2	Gets and sets the second membership function inflection coordinate parameter.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|          DifferencTwoSigmoidalMembershipFunction.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CDifferencTwoSigmoidalMembershipFunction func1(5,1,8,7);
CDifferencTwoSigmoidalMembershipFunction func2(5,4,5,7);
CDifferencTwoSigmoidalMembershipFunction func3(5,6,2,7);
//--- Create wrappers for membership functions
double DifferencTwoSigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double DifferencTwoSigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double DifferencTwoSigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"DifferencTwoSigmoidalMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"DifferencTwoSigmoidalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("DifferencTwoSigmoidalMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
```



```
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A1 (Get method)

Gets the first membership function slope ratio.

```
double A1()
```

Return Value

Slope ratio value.

A1 (Set method)

Sets the first membership function slope ratio.

```
void A1(
    const double a1    // slope ratio value
)
```

Parameters

a1

[in] Slope ratio value.

A2 (Get method)

Gets the second membership function slope ratio.

```
double A2()
```

Return Value

Slope ratio value.

A2 (Set method)

Sets the second membership function slope ratio.

```
void A2(
```



```
const double a2      // slope ratio value  
)
```

Parameters

a2

[in] Slope ratio value.

C1 (Get method)

Gets the first membership function inflection coordinate parameter.

```
double C1()
```

Return Value

Inflection coordinate value.

C1 (Set method)

Sets the first membership function inflection coordinate parameter.

```
void C1(  
    const double c1      // inflection coordinate value  
)
```

Parameters

c1

[in] Inflection coordinate value.

C2 (Get method)

Gets the second membership function inflection coordinate parameter.

```
double C2()
```

Return Value

Inflection coordinate value.

C2 (Set method)

Sets the second membership function inflection coordinate parameter.

```
void C2(  
    const double c2      // inflection coordinate value  
)
```

Parameters

c2

[in] Inflection coordinate value.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (  
    const double x      // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

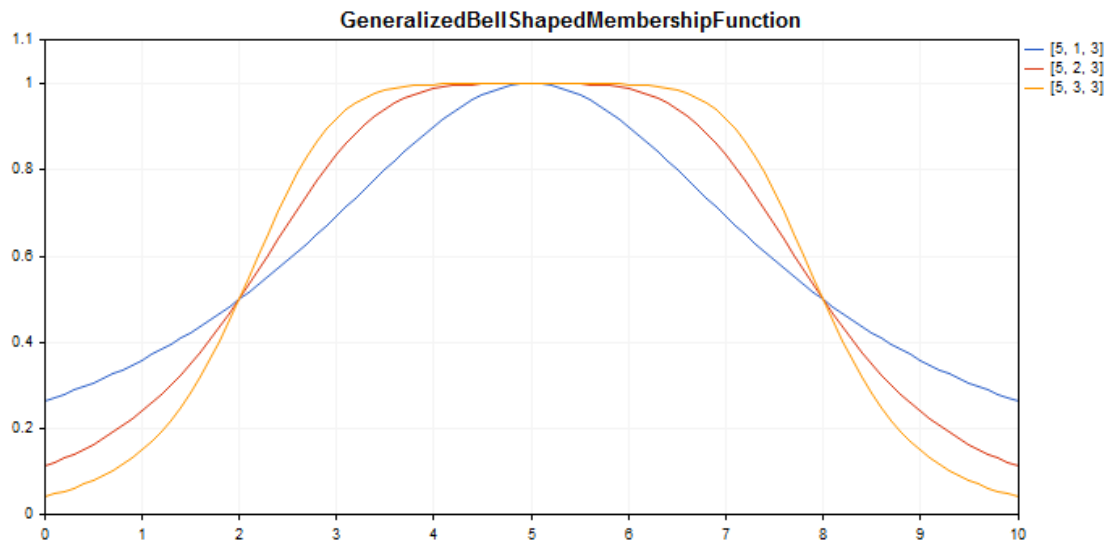
Membership function value.

CGeneralizedBellShapedMembershipFunction

Class for implementing a generalized bell-shaped membership function with A, B and C parameters.

Description

Generalized bell-shaped membership function shape is similar to Gaussian functions. The function is smooth and takes non-zero values along the entire definition area.



A [sample code](#) for plotting a chart is displayed below.

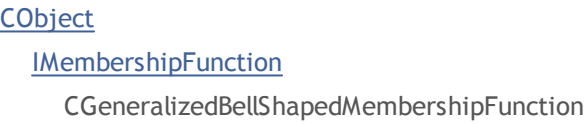
Declaration

```
class CGeneralizedBellShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy



Class methods

Class method	Description
A	Gets and sets the membership function concentration ratio.
B	Gets and sets the membership function slope ratio.

<u>C</u>	Gets and sets the membership function maximum coordinate.
<u>GetValue</u>	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|          GeneralizedBellShapedMembershipFunction.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CGeneralizedBellShapedMembershipFunction func1(5, 1, 3);
CGeneralizedBellShapedMembershipFunction func2(5, 2, 3);
CGeneralizedBellShapedMembershipFunction func3(5, 3, 3);
//--- Create wrappers for membership functions
double GeneralizedBellShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double GeneralizedBellShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double GeneralizedBellShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"GeneralizedBellShapedMembershipFunction",0,30,30,780,380))
{
graphic.Attach(0,"GeneralizedBellShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("GeneralizedBellShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
```



```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A (Get method)

Gets the membership function concentration ratio.

```
double A()
```

Return Value

Concentration ratio value.

A (Set method)

Sets the membership function concentration ratio.

```
void A(
    const double a    // concentration ratio value
)
```

Parameters

a

[in] Membership function concentration ratio.

B (Get method)

Gets the membership function slope ratio.

```
double B()
```

Return Value

Slope ratio value.

B (Set method)

Sets the membership function slope ratio.

```
void B(
    const double b    // slope ratio value
)
```



```
)
```

Parameters

b

[in] Membership function slope ratio.

C (Get method)

Gets the membership function maximum coordinate.

```
double C()
```

Return Value

Membership function maximum coordinate.

C (Set method)

Sets the membership function maximum coordinate.

```
void C(  
    const double c    // maximum coordinate value  
)
```

Parameters

c

[in] Membership function maximum coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const x    // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

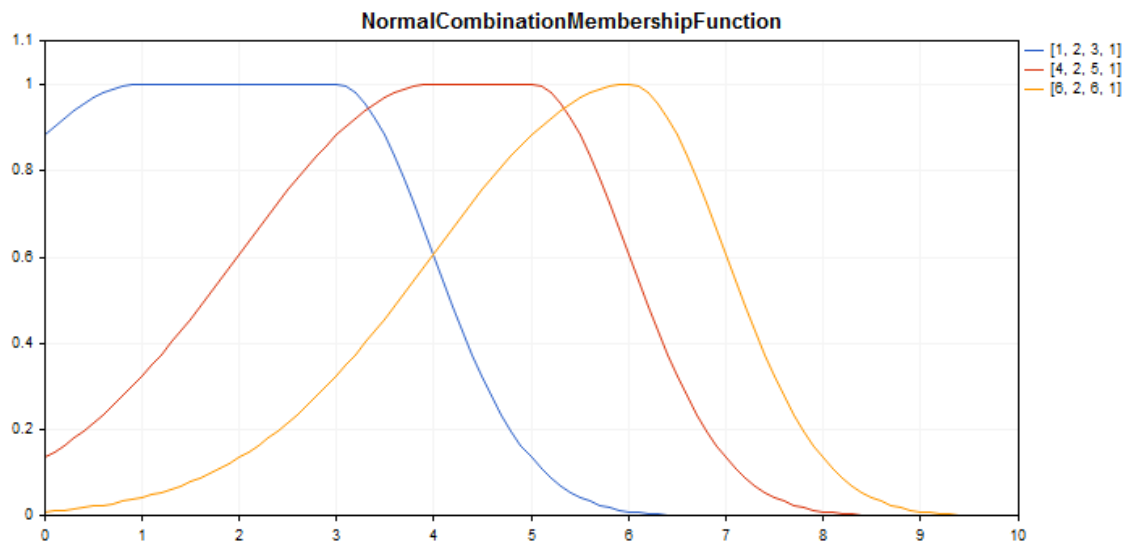
Membership function value.

CNormalCombinationMembershipFunction

Class for implementing a two-sided Gaussian membership function with the B1, B2, Sigma1 and Sigma2 parameters.

Description

The two-sided Gaussian membership function is formed using Gaussian distribution. It allows setting asymmetrical membership functions. The function is smooth and takes non-zero values along the entire definition area.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CNormalCombinationMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

[CObject](#)

[IMembershipFunction](#)

CNormalCombinationMembershipFunction

Class methods

Class method	Description
B1	Gets and sets the value of the first membership function center.

B2	Gets and sets the value of the second membership function center.
Sigma1	Gets and sets the first parameter of the membership function curvature.
Sigma2	Gets and sets the second parameter of the membership function curvature.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|          NormalCombinationMembershipFunction.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalCombinationMembershipFunction func1(1,2,3,1);
CNormalCombinationMembershipFunction func2(4,2,5,1);
CNormalCombinationMembershipFunction func3(6,2,6,1);
//--- Create wrappers for membership functions
double NormalCombinationMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalCombinationMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalCombinationMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"NormalCombinationMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"NormalCombinationMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("NormalCombinationMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(NormalCombinationMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[1,
graphic.CurveAdd(NormalCombinationMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[4,
graphic.CurveAdd(NormalCombinationMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[6,
```



```
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

B1 (Get method)

Gets the value of the first membership function center.

```
double B1()
```

Return Value

Value of the first membership function center.

B1 (Set method)

Sets the value of the first membership function center.

```
void B1(
    const double b1    // first center value
)
```

Parameters

b

[in] Value of the first membership function center.

B2 (Get method)

Gets the value of the second membership function center.

```
double B2()
```

Return Value

The value of the second membership function center.

B2 (Set method)

Sets the value of the second membership function center.

```
void B2(  
    const double b2      // second center value  
)
```

Parameters

b2

[in] Value of the second membership function center.

Sigma1 (Get method)

Gets the first parameter of the membership function curvature.

```
double Sigma1()
```

Return Value

The value of the first parameter of the membership function curvature.

Sigma1 (Set method)

Sets the value of the first parameter of the membership function curvature.

```
void Sigma1(  
    const double sigma1  // first curvature parameter value  
)
```

Parameters

sigma1

[in] The first parameter of the membership function curvature.

Sigma2 (Get method)

Gets the second parameter of the membership function curvature.

```
double Sigma2()
```

Return Value

The value of the second parameter of the membership function curvature.

Sigma2 (Set method)

Sets the value of the second parameter of the membership function curvature.

```
void Sigma2(  

```



```
const double sigma2 // second curvature parameter value
)
```

Parameters

sigma2

[in] The second curvature parameter of the membership function.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (
    const x // membership function argument
)
```

Parameters

x

[in] Membership function argument.

Return Value

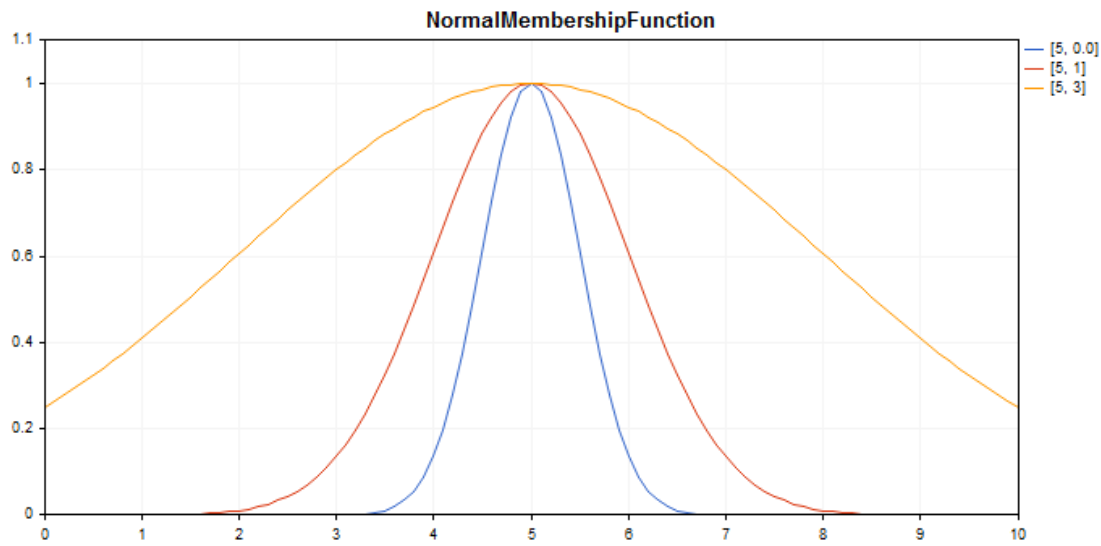
Membership function value.

CNormalMembershipFunction

Class for implementing a symmetrical Gaussian membership function with the B and Sigma parameters.

Description

The symmetrical Gaussian membership function is formed using Gaussian distribution. The function is smooth and takes non-zero values along the entire definition area.



A [sample code](#) for plotting a chart is displayed below.

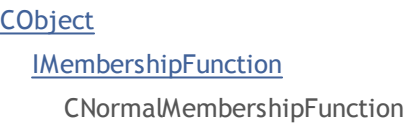
Declaration

```
class CNormalMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy



Class methods

Class method	Description
B	Gets and sets the membership function center.
Sigma	Gets and sets the parameter of the membership function curvature.

GetValue

Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     NormalMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalMembershipFunction func1(5,0.5);
CNormalMembershipFunction func2(5,1);
CNormalMembershipFunction func3(5,3);
//--- Create wrappers for membership functions
double NormalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"NormalMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"NormalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("NormalMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(NormalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[5, 0.0]");
graphic.CurveAdd(NormalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[5, 1]");
graphic.CurveAdd(NormalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[5, 3]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
```



```
graphic.YAxis().Min(0.0);  
graphic.YAxis().Max(1.1);  
graphic.YAxis().DefaultStep(0.2);  
/-- plot  
graphic.CurvePlotAll();  
graphic.Update();  
}
```

B (Get method)

Gets the membership function center.

```
double B()
```

Return Value

The value of the membership function center.

B (Set method)

Sets the value of the membership function center.

```
void B(  
    const double b    // function center value  
)
```

Parameters

b

[in] Value of the membership function center.

Sigma (Get method)

Gets the parameter of the membership function curvature.

```
double Sigma()
```

Return Value

The value of the membership function curvature parameter.

Sigma (Set method)

Sets the value of the membership function curvature parameter.

```
void Sigma(  
    const double sigma    // curvature parameter value  
)
```

Parameters

sigma

[in] Membership function curvature parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const double x    // argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

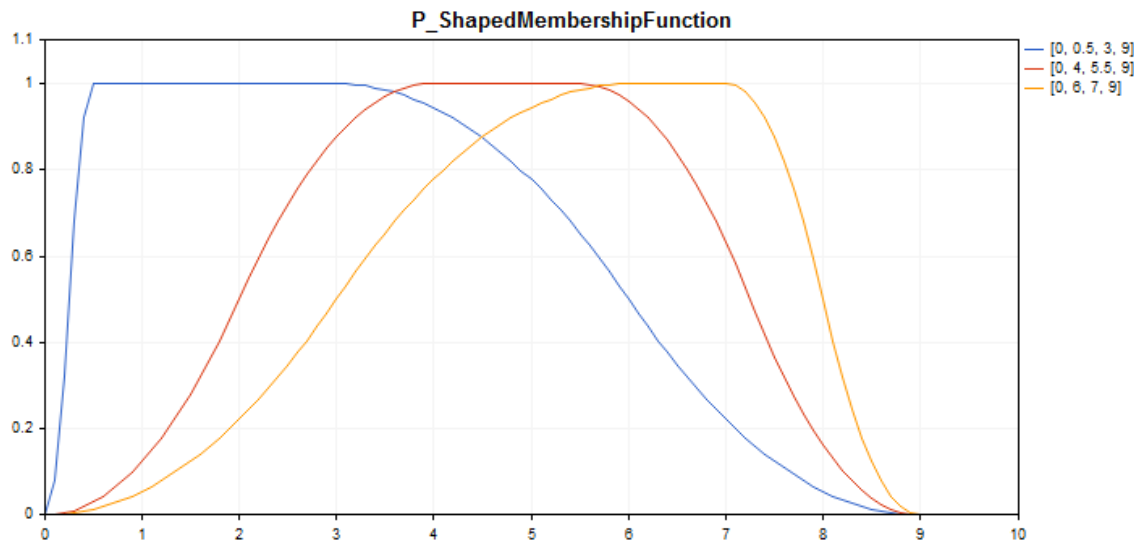
Membership function value.

CP_ShapedMembershipFunction

Class for implementing a pi-shaped membership function with the A, B, C and D parameters.

Description

The pi-shaped membership function has the form of a curvilinear trapezoid. The function is used to set asymmetric membership functions with a smooth transition from pessimistic to optimistic fuzzy number assessment.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CP_ShapedMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
 - [IMembershipFunction](#)
 - CP_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the fuzzy set beginning.
B	Gets and sets the first parameter of the fuzzy set core.

<u>C</u>	Gets and sets the second parameter of the fuzzy set core.
<u>D</u>	Gets and sets the parameter of the fuzzy set end.
<u>GetValue</u>	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     P_ShapedMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CP_ShapedMembershipFunction func1(0,0.5,3,9);
CP_ShapedMembershipFunction func2(0,4,5.5,9);
CP_ShapedMembershipFunction func3(0,6,7,9);
//--- Create wrappers for membership functions
double P_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double P_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"P_ShapedMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"P_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("P_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(P_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 0.5, 3,
graphic.CurveAdd(P_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 4, 5.5,
graphic.CurveAdd(P_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[0, 6, 7, 9]
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
```



```
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A (Get method)

Gets the parameter of the fuzzy set beginning.

```
double A()
```

Return Value

Parameter of the fuzzy set beginning.

A (Set method)

Sets the parameter of the fuzzy set beginning.

```
void A(
    const double a    // parameter of the fuzzy set beginning
)
```

Parameters

a

[in] Parameter of the fuzzy set beginning.

B (Get method)

Gets the first parameter of the fuzzy set core.

```
double B()
```

Return Value

The first parameter of the fuzzy set core.

B (Set method)

Sets the first parameter of the fuzzy set core.

```
void B(
    const double b    // value of the fuzzy set core first parameter
)
```


Parameters

b

[in] The first parameter of the fuzzy set core.

C (Get method)

Gets the second parameter of the fuzzy set core.

```
double C()
```

Return Value

The second parameter of the fuzzy set core.

C (Set method)

Sets the second parameter of the fuzzy set core.

```
void C(  
    const double c    // value of the fuzzy set core second parameter  
)
```

Parameters

c

[in] The second parameter of the fuzzy set core.

D (Get method)

Gets the parameter of the fuzzy set end.

```
double D()
```

Return Value

Value of the fuzzy set end parameter.

D (Set method)

Sets the parameter of the fuzzy set end.

```
void D(  
    const double d    // value of the fuzzy set end parameter  
)
```

Parameters

d

[in] Value of the fuzzy set end parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const double x  
)
```

Parameters

x

[in] membership function argument

Return Value

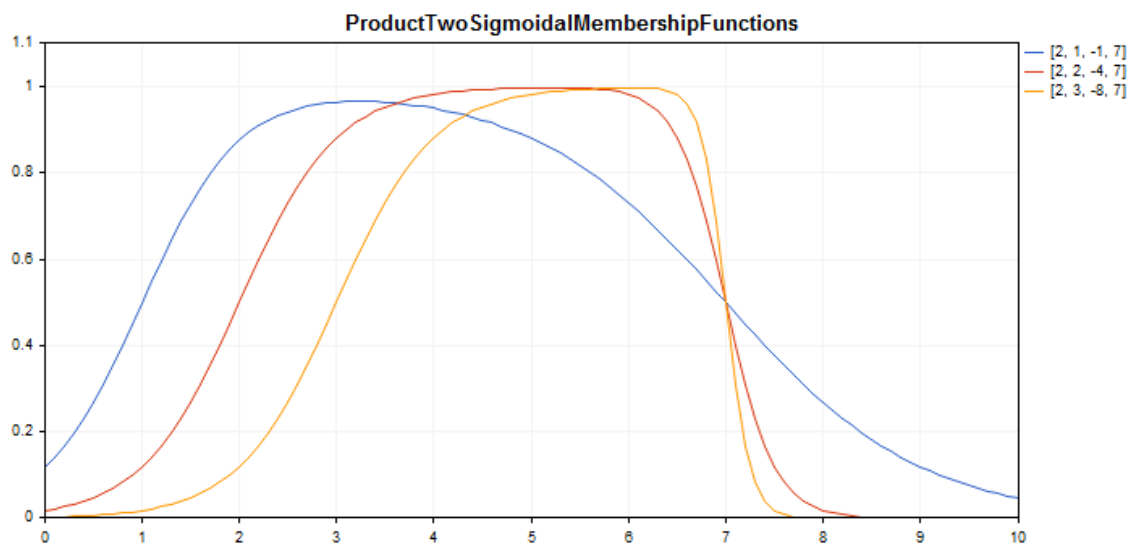
membership function value

CProductTwoSigmoidalMembershipFunction

Class for implementing the membership function in the form of a product of two sigmoid functions with the A1, A2, C1 and C2 parameters.

Description

A product of two sigmoid membership functions is applied for setting smooth asymmetric functions. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CProductTwoSigmoidalMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

[CObject](#)
[IMembershipFunction](#)
CProductTwoSigmoidalMembershipFunctions

Class methods

Class method	Description
A1	Gets and sets the first membership function slope ratio.

A2	Gets and sets the second membership function slope ratio.
C1	Gets the first membership function inflection coordinate parameter.
C2	Gets the second membership function inflection coordinate parameter.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|          ProductTwoSigmoidalMembershipFunctions.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CProductTwoSigmoidalMembershipFunctions func2(2,2,-4,7);
CProductTwoSigmoidalMembershipFunctions func3(2,3,-8,7);
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions1(double x) { return(func1.GetValue(x)); }
double ProductTwoSigmoidalMembershipFunctions2(double x) { return(func2.GetValue(x)); }
double ProductTwoSigmoidalMembershipFunctions3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"ProductTwoSigmoidalMembershipFunctions",0,30,30,780,380))
{
    graphic.Attach(0,"ProductTwoSigmoidalMembershipFunctions");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("ProductTwoSigmoidalMembershipFunctions");
graphic.BackgroundMainSize(16);
//--- create curve
```



```
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions1,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions2,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A1 (Get method)

Gets the first membership function slope ratio.

```
double A1()
```

Return Value

The first membership function slope ratio.

A1 (Set method)

Sets the first membership function slope ratio.

```
void A1(
    const double a1    // the first membership function slope ratio
)
```

Parameters

a1

[in] The first membership function slope ratio.

A2 (Get method)

Gets the second membership function slope ratio.

```
double A2()
```

Return Value

The second membership function slope ratio.

A2 (Set method)

Sets the second membership function slope ratio.

```
void A2(  
    const double a2      // the second membership function slope ratio  
)
```

Parameters

a2

[in] The second membership function slope ratio.

C1 (Get method)

Gets the first membership function inflection coordinate parameter.

```
double C1()
```

Return Value

The first membership function inflection coordinate.

C1 (Set method)

Sets the first membership function inflection coordinate.

```
void C1(  
    const double c1      // the first membership function inflection coordinate  
)
```

Parameters

c1

[in] The first membership function inflection coordinate.

C2 (Get method)

Gets the second membership function inflection coordinate parameter.

```
double C2()
```

Return Value

The second membership function inflection coordinate.

C2 (Set method)

Sets the second membership function inflection coordinate.

```
void C2(  
    const double c2      // the second membership function inflection coordinate  
)
```

Parameters

c2

[in] The second membership function inflection coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (
    const x      // membership function argument
)
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

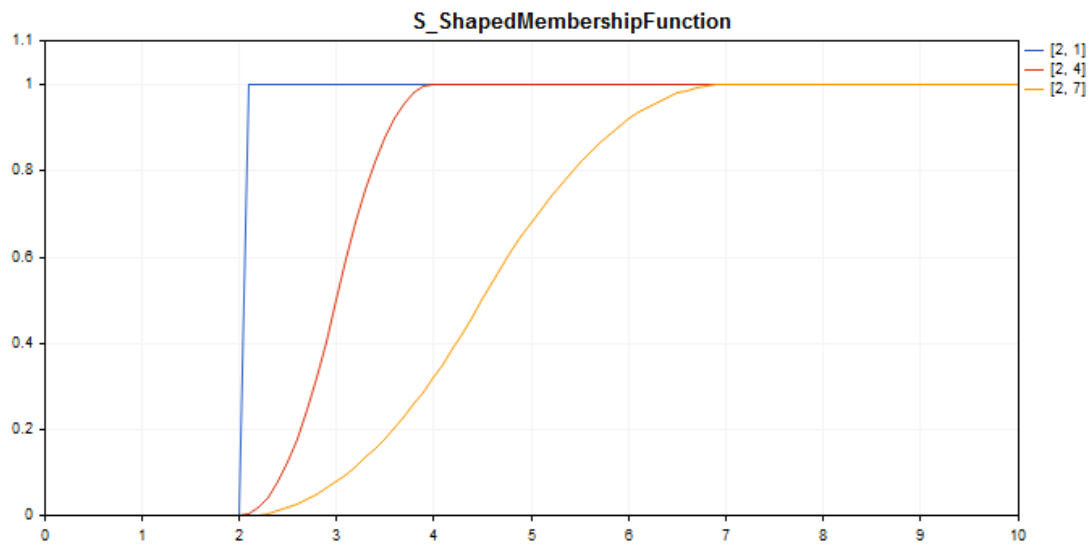
CS_ShapedMembershipFunction

Class for implementing an S-like membership function with the A and B parameters.

Description

The function sets an S-like two-parameter membership function. This is a non-decreasing function that takes values from 0 to 1. The A and B parameters define the interval, within which the function increases in non-linear trajectory from 0 to 1.

The function represents fuzzy sets of "very high" type (i.e. non-decreasing membership functions with saturation are set).



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CS_ShapedMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
- [IMembershipFunction](#)
- CS_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the increasing interval start.

<u>B</u>	Gets and sets the first parameter of the fuzzy set core.
<u>GetValue</u>	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     S_ShapedMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CS_ShapedMembershipFunction func1(2,1);
CS_ShapedMembershipFunction func2(2,4);
CS_ShapedMembershipFunction func3(2,7);
//--- Create wrappers for membership functions
double S_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double S_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double S_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"S_ShapedMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"S_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("S_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(S_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
graphic.CurveAdd(S_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
graphic.CurveAdd(S_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 7]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
```



```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A (Get method)

Gets the parameter of the increasing interval start.

```
double A()
```

Return Value

Increasing interval start parameter.

A (Set method)

Sets the parameter of the increasing interval start.

```
void A(
    const double a    // increasing interval start parameter
)
```

Parameters

a

[in] Increasing interval start parameter.

B (Get method)

Gets the first parameter of the fuzzy set core.

```
double B()
```

Return Value

The first parameter of the fuzzy set core.

B (Set method)

Sets the first parameter of the fuzzy set core.

```
void B(
    const double b    // the first parameter of the fuzzy set core
)
```

Parameters

b

[in] The first parameter of the fuzzy set core.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (
    const x      // membership function argument
)
```

Parameters

x

[in] Membership function argument.

Return Value

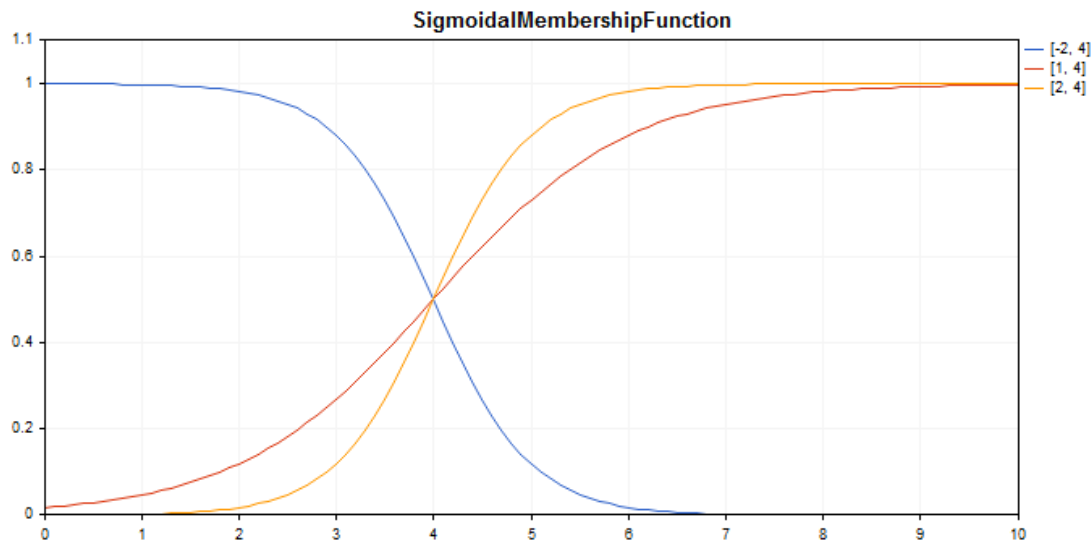
Membership function value.

CSigmoidalMembershipFunction

Class for implementing a sigmoid membership function with the A and C parameters.

Description

The sigmoid function is applied when setting monotonous membership functions. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CSigmoidalMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
- [IMembershipFunction](#)
- CSigmoidalMembershipFunction

Class methods

Class method	Description
A	Gets and sets the membership function slope ratio.
C	Gets and sets the membership function inflection coordinate parameter.

GetValue

Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                               SigmoidalMembershipFunction.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
/-- create membership functions
CSigmoidalMembershipFunction func1(-2, 4);
CSigmoidalMembershipFunction func2(1, 4);
CSigmoidalMembershipFunction func3(2, 4);
/-- create wrappers for membership functions
double SigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double SigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double SigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
/-- create graphic
CGraphic graphic;
if(!graphic.Create(0,"SigmoidalMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"SigmoidalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("SigmoidalMembershipFunction");
graphic.BackgroundMainSize(16);
/-- create curve
graphic.CurveAdd(SigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[-2, 4]");
graphic.CurveAdd(SigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[1, 4]");
graphic.CurveAdd(SigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
/-- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
/-- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
```



```
graphic.YAxis().Min(0.0);  
graphic.YAxis().Max(1.1);  
graphic.YAxis().DefaultStep(0.2);  
/-- plot  
graphic.CurvePlotAll();  
graphic.Update();  
}
```

A (Get method)

Gets the membership function slope ratio.

```
double A()
```

Return Value

The membership function slope ratio.

A (Set method)

Sets the membership function slope ratio.

```
void A(  
    const double a    // the first membership function slope ratio  
)
```

Parameters

a

[in] Membership function slope ratio.

C (Get method)

Gets the membership function inflection coordinate parameter.

```
double C()
```

Return Value

Membership function inflection coordinate.

C (Set method)

Sets the membership function inflection coordinate.

```
void C(  
    const double c    // membership function inflection coordinate  
)
```

Parameters

c

[in] The membership function inflection coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const x      // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

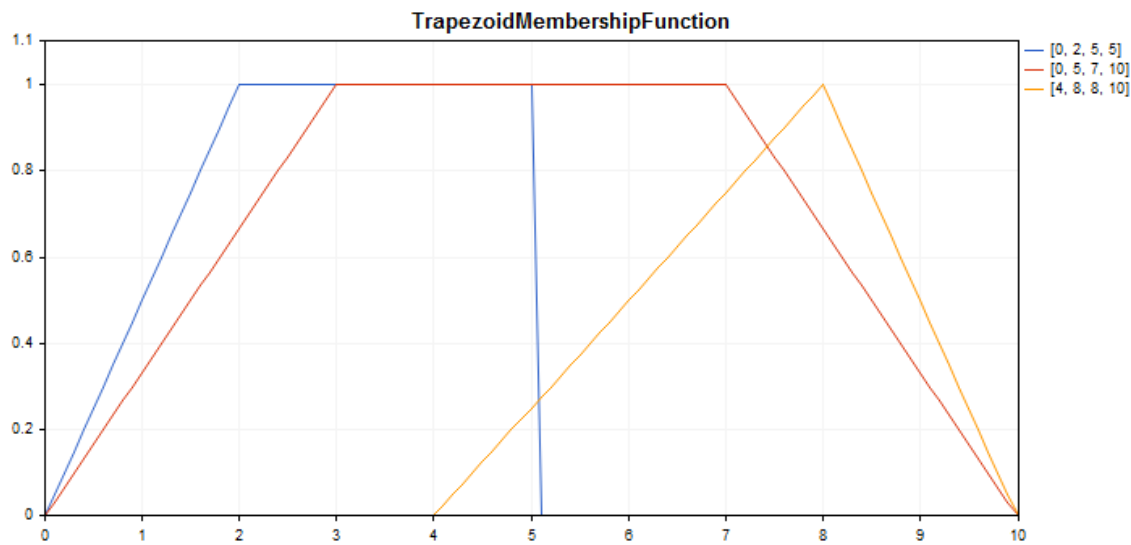
CTrapezoidMembershipFunction

Class for implementing a trapezoidal membership function with the X1, X2, X3 and X4 parameters.

Description

The function is formed using piecewise linear approximation. This is a generalization of the triangular function allowing you to assign a fuzzy set core as an interval. Such a membership function makes it possible to conveniently interpret optimistic/pessimistic assessments.

The function is used to set asymmetric membership functions of the variables with their most critical values defined within a certain interval.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CTrapezoidMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

[CObject](#)
[IMembershipFunction](#)
CTrapezoidMembershipFunction

Class methods

Class method	Description
X1	Gets and sets the value of the first point on the X axis.

X2	Gets and sets the value of the second point on the X axis.
X3	Gets and sets the value of the third point on the X axis.
X4	Gets and sets the value of the fourth point on the X axis.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     TrapezoidMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTrapezoidMembershipFunction func1(0,2,5,5);
CTrapezoidMembershipFunction func2(0,3,7,10);
CTrapezoidMembershipFunction func3(4,8,8,10);
//--- Create wrappers for membership functions
double TrapezoidMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TrapezoidMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TrapezoidMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"TrapezoidMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"TrapezoidMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("TrapezoidMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(TrapezoidMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5, 5");
graphic.CurveAdd(TrapezoidMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 7, 1");
graphic.CurveAdd(TrapezoidMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[4, 8, 8, 1");
```



```
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

X1 (Get method)

Gets the value of the first point on the X axis.

```
double X1()
```

Return Value

The value of the first point on the X axis.

X1 (Set method)

Sets the value of the first point on the X axis.

```
void X1(
    const double x1    // value of the first point on the X axis
)
```

Parameters

x1

[in] The value of the first point on the X axis.

X2 (Get method)

Gets the value of the second point on the X axis.

```
double X2()
```

Return Value

The value of the second point on the X axis.

X2 (Set method)

Sets the value of the second point on the X axis.

```
void X2(
```



```
const double x2      // value of the second point on the X axis
)
```

Parameters

x2

[in] The value of the second point on the X axis.

X3 (Get method)

Gets the value of the third point on the X axis.

```
double X3()
```

Return Value

The value of the third point on the X axis.

X3 (Set method)

Sets the value of the third point on the X axis.

```
void X3(
    const double x3      // value of the third point on the X axis
)
```

Parameters

x3

[in] The value of the third point on the X axis.

X4 (Get method)

Gets the value of the fourth point on the X axis.

```
double X4()
```

Return Value

The value of the fourth point on the X axis.

X4 (Set method)

Sets the value of the fourth point on the X axis.

```
void X4(
    const double x4      // value of the fourth point on the X axis
)
```

Parameters

x4

[in] The value of the fourth point on the X axis.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const x      // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

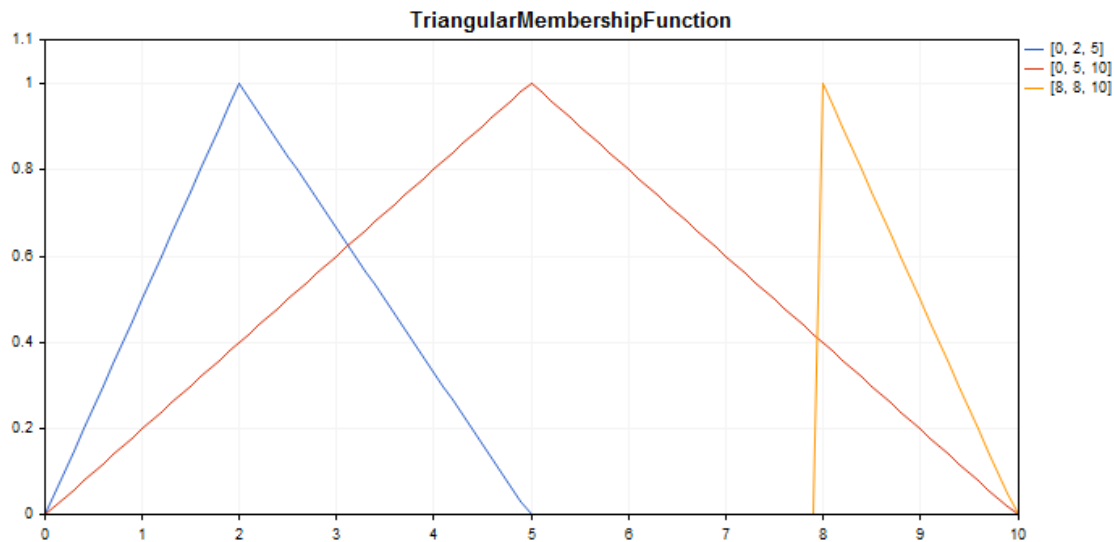
Membership function value.

CTriangularMembershipFunction

Class for implementing a triangle membership function with the X1, X2 and X3 parameters.

Description

The function sets a membership function in the form of a triangle. This is a simple and most frequently applied membership function.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CTriangularMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

CObject

IMembershipFunction

CTriangularMembershipFunction

Class methods

Class method	Description
<u>X1</u>	Gets the value of the first point on the X axis.
<u>X2</u>	Gets the value of the second point on the X axis.
<u>X3</u>	Gets the value of the third point on the X axis.

ToNormalMF	Converts a triangle membership function into a Gaussian one.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                               TriangularMembershipFunction.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+

#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTriangularMembershipFunction func1(0,2,5);
CTriangularMembershipFunction func2(0,5,10);
CTriangularMembershipFunction func3(8,8,10);
//--- Create wrappers for membership functions
double TriangularMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TriangularMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TriangularMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"TriangularMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"TriangularMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("TriangularMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(TriangularMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5]");
graphic.CurveAdd(TriangularMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 10]");
graphic.CurveAdd(TriangularMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[8, 8, 10]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
```



```
graphic.XAxis().Min(0.0);  
graphic.XAxis().Max(10.0);  
graphic.XAxis().DefaultStep(1.0);  
//--- sets the Y-axis properties  
graphic.YAxis().AutoScale(false);  
graphic.YAxis().Min(0.0);  
graphic.YAxis().Max(1.1);  
graphic.YAxis().DefaultStep(0.2);  
//--- plot  
graphic.CurvePlotAll();  
graphic.Update();  
}
```

X1 (Get method)

Gets the value of the first point on the X axis.

```
double X1()
```

Return Value

The value of the first point on the X axis.

X1 (Set method)

Sets the value of the first point on the X axis.

```
void X1(  
    const double x1    // value of the first point on the X axis  
)
```

Parameters

x1

[in] The value of the first point on the X axis.

X2 (Get method)

Gets the value of the second point on the X axis.

```
double X2()
```

Return Value

The value of the second point on the X axis.

X2 (Set method)

Sets the value of the second point on the X axis.

```
void X2(  
    const double x2    // value of the second point on the X axis  
)
```


Parameters

x2

[in] The value of the second point on the X axis.

X3 (Get method)

Gets the value of the third point on the X axis.

```
double X3 ()
```

Return Value

The value of the third point on the X axis.

X3 (Set method)

Sets the value of the third point on the X axis.

```
void X3 (
    const double x3      // value of the third point on the X axis
)
```

Parameters

x3

[in] The value of the third point on the X axis.

ToNormalMF

Converts a triangle membership function into a Gaussian one.

```
CNormalMembershipFunction* ToNormalMF ()
```

Return Value

The pointer to a [Gaussian membership function](#).

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (
    const x      // membership function argument
)
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

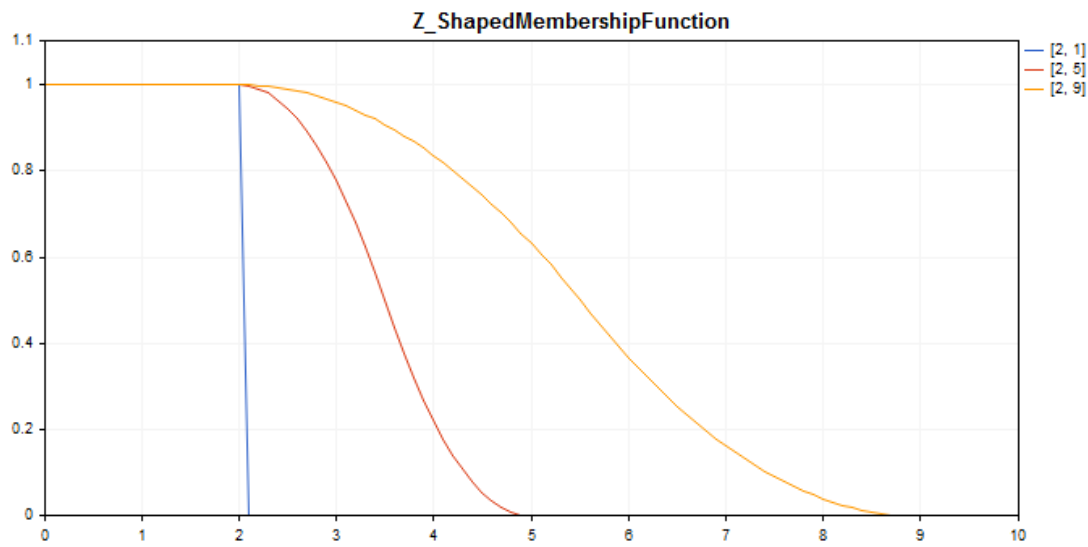
CZ_ShapedMembershipFunction

Class for implementing a z-like membership function with the A and B parameters.

Description

The function sets a z-like two-parameter membership function. This is a non-increasing membership function that takes values from 1 to 0. The function parameters define an interval, within which the function decreases in non-linear trajectory from 1 to 0.

The function represents fuzzy sets of "very low" type. In other words, it sets non-increasing membership functions with saturation.



A [sample code](#) for plotting a chart is displayed below.

Declaration

```
class CZ_ShapedMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- [CObject](#)
- [IMembershipFunction](#)
- CZ_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the decreasing interval start.

<u>B</u>	Gets and sets the parameter of the decreasing interval end.
<u>GetValue</u>	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Example

```
//+-----+
//|                                     Z_ShapedMembershipFunction.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CZ_ShapedMembershipFunction func1(2,1);
CZ_ShapedMembershipFunction func2(2,5);
CZ_ShapedMembershipFunction func3(2,9);
//--- Create wrappers for membership functions
double Z_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double Z_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double Z_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"Z_ShapedMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"Z_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("Z_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(Z_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
graphic.CurveAdd(Z_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 5]");
graphic.CurveAdd(Z_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 9]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
```



```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

A (Get method)

Gets the parameter of the decreasing interval start.

```
double A()
```

Return Value

Decreasing interval start parameter.

A (Set method)

Sets the parameter of the decreasing interval start.

```
void A(
    const double a    // decreasing interval start parameter
)
```

Parameters

a

[in] Decreasing interval start parameter.

B (Get method)

Gets the parameter of the decreasing interval end.

```
double B()
```

Return Value

Decreasing interval end parameter.

B (Set method)

Sets the parameter of the decreasing interval end.

```
void B(
    const double b    // decreasing interval end parameter
)
```

Parameters

b

[in] Decreasing interval end parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue (  
    const x      // membership function argument  
)
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

IMembershipFunction

Basic class for all membership function classes.

Declaration

```
class CZ_ShapedMembershipFunction : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

[CObject](#)
IMembershipFunction

Direct descendants

[CCompositeMembershipFunction](#), [CConstantMembershipFunction](#),
[CDifferencTwoSigmoidalMembershipFunction](#), [CGeneralizedBellShapedMembershipFunction](#),
[CNormalCombinationMembershipFunction](#), [CNormalMembershipFunction](#),
[CP_ShapedMembershipFunction](#), [CProductTwoSigmoidalMembershipFunctions](#),
[CS_ShapedMembershipFunction](#), [CSigmoidalMembershipFunction](#), [CTrapezoidMembershipFunction](#),
[CTriangularMembershipFunction](#), [CZ_ShapedMembershipFunction](#)

Class methods

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

GetValue

Calculates the value of the membership function by a specified argument.

```
double GetValue(  
    const x // membership function argument  
)
```

Parameters

x
[in] Membership function argument.

Return Value

Membership function value.

Fuzzy systems rules

A **fuzzy system** (fuzzy logical inference system) is a receipt of conclusion in the form of a fuzzy set corresponding to the current values of the inputs with the use of a set of **fuzzy rules** and fuzzy operations.

The **fuzzy rules** determine the relationship between inputs and outputs of an examined object. Amount of rules in the system is unlimited. The generalized format of fuzzy rules is as follows:

if rule condition, then rule conclusion.

Rule condition describes the current state of the object. *Rule conclusion* describes how the condition affects the object.

Class of rules for fuzzy systems	Description
<u>CMamdaniFuzzyRule</u>	Class for implementing a fuzzy logic rule for the Mamdani algorithm
<u>CSugenoFuzzyRule</u>	Class for implementing a fuzzy logic rule for the Sugeno algorithm
<u>CSingleCondition</u>	The class sets a fuzzy condition expressed by "Fuzzy variable – Fuzzy term" pair.
<u>CConditions</u>	Class defines a set of fuzzy conditions connected to each other by an operator.
<u>CGenericFuzzyRule</u>	Base class for implementing the both types of fuzzy rules.

CMamdaniFuzzyRule

Mamdani-type fuzzy inference – one of the two basic types of fuzzy systems. Output variable values are set using fuzzy terms.

Description

Fuzzy logic rule for the Mamdani algorithm can be described as follows:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y \text{ is } d)(W)$$

where:

- X = (X1, X2, X3 ... Xn) – vector of input variables;
- Y – output variable;
- a = (a1, a2, a3 ... an) – vector of input variable values;
- d – output variable value;
- W – rule weight.

Declaration

```
class CMamdaniFuzzyRule : public CGenericFuzzyRule
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy



Class methods

Class method	Description
Conclusion	Gets and sets the Mamdani fuzzy rule conclusion
Weight	Gets and sets the Mamdani fuzzy rule weight

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CGenericFuzzyRule

[Condition](#), [Condition](#), [CreateCondition](#), [CreateCondition](#), [CreateCondition](#)

Conclusion (Get method)

Gets the Mamdani fuzzy rule conclusion

```
CSingleConditon* Conclusion()
```

Return Value

Conclusion of a Mamdani fuzzy rule.

Conclusion (Set method)

Sets the Mamdani fuzzy rule conclusion.

```
void Conclusion(  
    CSingleConditon* value    // conclusion of a Mamdani fuzzy rule  
)
```

Parameters

value

[in] Conclusion of a Mamdani fuzzy rule.

Weight (Get method)

Gets the Mamdani fuzzy rule weight.

```
double Weight()
```

Return Value

Mamdani fuzzy rule weight.

Weight (Set method)

Sets the Mamdani fuzzy rule weight.

```
void Weight(  
    const double value    // weight of a Mamdani fuzzy rule  
)
```

Parameters

value

[in] Weight of a Mamdani fuzzy rule.

CSugenoFuzzyRule

Sugeno-type fuzzy inference – one of the two basic types of fuzzy systems. Output variable values are set as a linear combination of input variables.

Description

Unlike the Mamdani rule, an input variable value is set by a linear function from entries rather than by a fuzzy term. Fuzzy logic rule for the Sugeno algorithm can be described as follows:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y = b_0 + b_1 \cdot X_1 + b_2 \cdot X_2 + \dots + b_n \cdot X_n)(W)$$

where:

- X = (X1, X2, X3 ... Xn) – vector of input variables;
- Y – output variable;
- a = (a1, a2, a3 ... an) – vector of input variable values;
- b = (b1, b2, b3 ... bn) – free term ratio in the linear function for an output value
- W – rule weight.

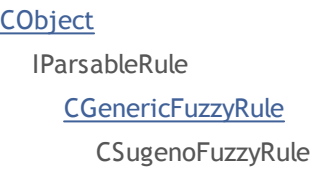
Declaration

```
class CSugenoFuzzyRule : public CGenericFuzzyRule
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy



Class methods

Class method	Description
Conclusion	Gets and sets the Sugeno fuzzy rule conclusion

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CGenericFuzzyRule

[Condition](#), [Condition](#), [CreateCondition](#), [CreateCondition](#), [CreateCondition](#)

Conclusion (Get method)

Gets the Sugeno fuzzy rule conclusion.

```
CSingleConditon* Conclusion()
```

Return Value

Sugeno fuzzy rule conclusion.

Conclusion (Set method)

Sets the Sugeno fuzzy rule conclusion.

```
void Conclusion(  
    CSingleCondition* value    // conclusion of a Sugeno fuzzy rule  
)
```

Parameters

value

[in] Conclusion of a Sugeno fuzzy rule.

CSingleCondition

The class sets a fuzzy condition expressed by "Fuzzy variable – Fuzzy term" pair.

Description

According to a fuzzy condition, one variable corresponds to one term. A fuzzy condition can be described by the following expression: X is a ,

where:

- X is a fuzzy variable;
- a is a fuzzy variable value (fuzzy term).

Declaration

```
class CSingleCondition : public ICondition
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject

ICondition

CSingleCondition

Direct descendants

CFuzzyCondition

Class methods

Class method	Description
Not	Gets and sets the flag indicating whether it is necessary to apply negation to this condition.
Term	Gets and sets a fuzzy term for this condition.
Var	Gets and sets a fuzzy variable for this condition.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Not (Get method)

Gets the flag indicating whether it is necessary to apply negation to this condition.

```
bool Not()
```


Return Value

The flag value.

Not (Set method)

Sets the flag indicating whether it is necessary to apply negation to this condition.

```
void Not(  
    bool not      // flag value  
)
```

Parameters

not

[in] Flag value.

Term (Get method)

Gets a fuzzy term for the given condition.

```
INamedValue* Term()
```

Return Value

A fuzzy term for the given condition.

Term (Set method)

Sets a fuzzy term for the given condition.

```
void Term(  
    INamedValue*& value      // fuzzy term for the given condition  
)
```

Parameters

value

[in] A fuzzy term for the given condition.

Var (Get method)

Gets a fuzzy variable for the given condition.

```
INamedVariable* Var()
```

Return Value

A fuzzy variable for the given condition.

Var (Set method)

Sets a fuzzy variable for the given condition


```
void Var(  
    INamedVariable*& value    // a fuzzy variable for the given condition  
)
```

Parameters

value

[in] fuzzy variable.

CConditions

Class defines a set of fuzzy conditions connected to each other by an operator.

Description

A set of fuzzy conditions connected to each other by an operator may be described as follows:

$$(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n)$$

where:

- $X = (X_1, X_2, X_3 \dots X_n)$ – vector of input variables;
- $a = (a_1, a_2, a_3 \dots a_n)$ – vector of input variable values.

In this example, the *and* operator is used. Besides, the *or* operator is available in this class.

Declaration

```
class CConditions : public ICondition
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject

ICondition

CConditions

Class methods

Class method	Description
<u>ConditionsList</u>	Gets the list of all conditions
<u>Not</u>	Gets and sets the flag indicating whether it is necessary to apply negation to these conditions
<u>Op</u>	Gets and sets a type of the conditions bundle operator.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

ConditionsList

Gets the list of all conditions.

```
CList* ConditionsList()
```


Return Value

List of all conditions.

Not (Get method)

Gets the flag indicating whether it is necessary to apply negation to these conditions.

```
bool Not()
```

Return Value

The flag value.

Not (Set method)

Sets the flag indicating whether it is necessary to apply negation to these conditions

```
void Not(  
    bool not    // flag value  
)
```

Parameters

not

[in] Flag value.

Op (Get method)

Gets a type of the conditions bundle operator. The *and* and *or* operators are available.

```
OperatorType Op()
```

Return Value

Type of the conditions bundle operator.

Op (Set method)

Set the conditions bundle operator. The *and* and *or* operators are available.

```
void Op(  
    OperatorType op    // type of the conditions bundle operator  
)
```

Parameters

op

[in] Type of the conditions bundle operator.

CGenericFuzzyRule

Base class for both types of fuzzy rules.

Declaration

```
class CGenericFuzzyRule : public IParsableRule
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject

IParsableRule

CGenericFuzzyRule

Direct descendants

CMamdaniFuzzyRule, CSugenoFuzzyRule

Class methods

Class method	Description
<u>Conclusion</u>	Gets and sets the fuzzy rule conclusion
<u>Condition</u>	Gets and sets the 'if' condition (set of conditions) for a fuzzy rule
<u>CreateCondition</u>	Creates a condition for a fuzzy rule by specified parameters

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Conclusion (Get method)

Gets the conclusion of a fuzzy rule.

```
CSingleConditon* Conclusion()
```

Return Value

Conclusion of a fuzzy rule.

Conclusion (Set method)

Sets the fuzzy rule conclusion.

```
virtual void Conclusion(
    CSingleConditon* value    // conclusion of a fuzzy rule
```



```
)
```

Parameters*value*

[in] Conclusion of a fuzzy rule.

Condition (Get method)

Gets the 'if' condition (set of conditions) for a fuzzy rule.

```
CConditons* Condition()
```

Return Value

Fuzzy condition (set of conditions).

Condition (Set method)

Sets the 'if' condition (set of conditions) for a fuzzy rule.

```
void Condition(
    CConditons* value    // 'if' condition (set of conditions) for a fuzzy rule
)
```

Parameters*value*

[in] Fuzzy condition (set of conditions).

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var,    // fuzzy variable
    CFuzzyTerm* term,      // fuzzy term
)
```

Parameters*var*

[in] Fuzzy variable.

term

[in] Fuzzy term.

Return Value

Fuzzy rule status.

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var,      // fuzzy variable
    CFuzzyTerm* term,        // fuzzy term
    bool not,                // flag indicating whether it is necessary to apply negation
)
```

Parameters

var

[in] Fuzzy variable.

term

[in] Fuzzy term.

not

[in] Flag indicating whether it is necessary to apply negation to a condition.

Return Value

Fuzzy rule status.

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var,      // fuzzy variable
    CFuzzyTerm* term,        // fuzzy term
    bool not,                // flag indicating whether it is necessary to apply negation
    HedgeType hedge          // condition bundle type
)
```

Parameters

var

[in] Fuzzy variable.

term

[in] Fuzzy term.

not

[in] Flag indicating whether it is necessary to apply negation to a condition.

hedge

[in] Condition bundle type.

Return Value

Fuzzy rule status.

Fuzzy systems variables

Fuzzy (linguistic) variables **are applied in fuzzy systems**. These are the variables whose values are words or word combinations in a natural or artificial language.

Linguistic variables comprise fuzzy sets. The nature and number of fuzzy variables change for each certain task when defining fuzzy sets.

Class	Description
<u>CFuzzyVariable</u>	Class for creating general fuzzy variables.
<u>CSugenoVariable</u>	Class for creating fuzzy Sugeno-type variables.

CFuzzyVariable

Class for creating general fuzzy variables.

Description

Here, a fuzzy variable is created with the following parameters:

- maximum variable value;
- minimum variable value;
- fuzzy variable name;
- term set (set of all possible values, which a linguistic variable is capable of receiving).

Declaration

```
class CFuzzyVariable : public CNamedVariableImpl
```

Title

```
#include <Math\Fuzzy\fuzzyvariable.mqh>
```

Inheritance hierarchy

CObject

INamedValue

INamedVariable

CNamedVariableImpl

CFuzzyVariable

Class methods

Class method	Description
<u>AddTerm</u>	Adds a single fuzzy term to a fuzzy variable.
<u>GetTermByName</u>	Gets a fuzzy term by a specified name.
<u>Max</u>	Gets and sets a maximum value for a fuzzy variable.
<u>Min</u>	Gets and sets a minimum value for a fuzzy variable.
<u>Terms</u>	Gets and sets a list of fuzzy terms for the given fuzzy variable.
<u>Values</u>	Gets and sets a list of fuzzy terms for the given fuzzy variable.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CNamedVariableImpl

Name, Name

AddTerm

Adds a single fuzzy term to a fuzzy variable.

```
void AddTerm(  
    CFuzzyTerm* & term    // fuzzy term  
)
```

Parameters

term

[in] Fuzzy term.

GetTermByName

Gets a fuzzy term by a specified name.

```
CFuzzyTerm* GetTermByName(  
    const string name    // fuzzy term name  
)
```

Parameters

name

[in] Fuzzy term name.

Return Value

Fuzzy term with a specified name.

Max (Get method)

Gets the maximum value for a fuzzy variable.

```
double Max()
```

Return Value

Maximum value for a fuzzy variable.

Max (Set method)

Sets the maximum value for a fuzzy variable.

```
void Max(  
    const double max    // maximum value for a fuzzy variable  
)
```

Parameters

max

[in] Maximum value for a fuzzy variable.

Min (Get method)

Gets the minimum value for a fuzzy variable.

```
double Min()
```

Return Value

Minimum value for a fuzzy variable.

Max (Set method)

Sets the minimum value for a fuzzy variable.

```
void Min(  
    const double min    // minimum value for a fuzzy variable  
)
```

Parameters

min

[in] Minimum value for a fuzzy variable.

Terms (Get method)

Gets a list of fuzzy terms for the given fuzzy variable.

```
CList* Terms()
```

Return Value

List of fuzzy terms for the given fuzzy variable.

Terms (Set method)

Sets a list of fuzzy terms for the given fuzzy variable.

```
void Terms(  
    CList*& terms    // list of fuzzy terms for the given variable  
)
```

Parameters

terms

[in] list of fuzzy terms for the given fuzzy variable.

Values

Gets a list of fuzzy terms for the given fuzzy variable.


```
CList* Values()
```

Return Value

List of fuzzy terms for the given variable.

CSugenoVariable

Class for creating fuzzy Sugeno-type variables.

Description

Fuzzy Sugeno-type variable is different from the general linguistic variable since it is not set by a term set but by a set of linear functions.

Declaration

```
class CSugenoVariable : public CNamedVariableImpl
```

Title

```
#include <Math\Fuzzy\sugenovvariable.mqh>
```

Inheritance hierarchy

CObject

INamedValue

INamedVariable

CNamedVariableImpl

CSugenoVariable

Class methods

Class method	Description
Functions	Gets the list of linear functions of the fuzzy Sugeno variable.
GetFuncByName	Gets the linear function by a specified name.
Values	Gets the list of linear functions of the fuzzy Sugeno variable.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CNamedVariableImpl

Name, Name

Functions

Gets the list of linear functions of the fuzzy Sugeno variable.

```
CList* Functions()
```

Return Value

List of linear functions.

GetFuncByName

Gets the linear function by a specified name.

```
ISugenoFunction* GetFuncByName (  
    const string name // linear function name  
)
```

Parameters

name

[in] Linear function name.

Return Value

Linear function with a specified name.

Values

Gets the list of linear functions of the fuzzy Sugeno variable.

```
CList* Values ()
```

Return Value

List of linear functions of the fuzzy Sugeno variable.

CFuzzyTerm (fuzzy terms)

Class for implementing fuzzy terms.

Description

A **term** is any element of a term set. A term is defined by two components:

- fuzzy term name;
- membership function.

Declaration

```
class CFuzzyTerm : public CNamedValueImpl
```

Title

```
#include <Math\Fuzzy\fuzzyterm.mqh>
```

Inheritance hierarchy

CObject

INamedValue

CNamedValueImpl

CFuzzyTerm

Class methods

Class method	Description
MembershipFunction	Gets a membership function for the fuzzy term.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CNamedValueImpl

Name, Name

MembershipFunction

Gets a membership function for the fuzzy term.

```
IMembershipFunction* MembershipFunction()
```

Return Value

Membership function

Fuzzy systems

Fuzzy system (or *fuzzy model*) is a mathematical model whose calculation is based on fuzzy logic. Construction of such models is applicable when the subject of study has a weak formalization and its exact mathematical description is too complex or unknown.

The progress of a model construction can be divided into three main stages:

1. Definition of input and output characteristics of a model.
2. Building a knowledge base.
3. Selecting one of the methods of fuzzy inference (Mamdani and Sugeno).

The first stage directly effects the consequent two and determines the future operation of the model.

A **knowledge base** (*rule base*) is a set of fuzzy rules of "if, then" type that define the relationship between inputs and outputs of the examined object.

Rule condition describes the current state of the object, and **rule conclusion** — how this condition affects the object.

There can be two types of terms and conclusions for each rule:

1. simple (link to [Csinglecond](#)) — includes one fuzzy variable;
2. complex (link to [Cconditions](#)) — includes several fuzzy variables.

Each rule in the system has its **weight** — importance of a rule in the model. Weighting factors are assigned to a rule within range [0, 1].

Depending on the created knowledge base, the system of fuzzy inference is determined for a model.

Fuzzy logical inference is a receipt of conclusion in form of a fuzzy set corresponding to the current value of the inputs with use of knowledge base and fuzzy operations. The two main types of fuzzy inference are Mamdani and Sugeno.

Mamdani system

Output variable values in the Mamdani system are set using fuzzy terms.

Description

Fuzzy logic rule for the Mamdani algorithm can be described as follows:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y \text{ is } d)(W)$$

where:

- $X = (X_1, X_2, X_3 \dots X_n)$ – vector of input variables;
- Y – output variable;
- $a = (a_1, a_2, a_3 \dots a_n)$ – vector of input variable values;
- d – output variable value;
- W – rule weight.

Class methods

Class method	Description
AggregationMethod	Sets the type of conditions aggregation
Calculate	Calculates a fuzzy inference for the system
DefuzzificationMethod	Sets defuzzification method type
EmptyRule	Creates an empty fuzzy Mamdani rule based on the current system
ImplicationMethod	Sets a type of the system implication operator
Output	Gets the list of fuzzy Mamdani output variables.
OutputByName	Gets a fuzzy Mamdani output variable by a specified name.
ParseRule	Creates a fuzzy Mamdani rule based on a specified line.
Rules	Returns the list of fuzzy Mamdani rules.

Methods inherited from class CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

AggregationMethod

Sets the type of conditions aggregation method.

```
void AggregationMethod(
    AggregationMethod value    // aggregation method type
```



```
)
```

Parameters

value

[in] Type of conditions aggregation method.

Calculate

Calculates a fuzzy inference for the system

```
CList* Calculate(  
    CList* inputValues    // input data  
)
```

Parameters

inputValues

[in] Input data for calculation.

Return Value

Calculation result.

DefuzzificationMethod

Sets defuzzification method type.

```
void DefuzzificationMethod(  
    DefuzzificationMethod value    // defuzzification method type.  
)
```

Parameters

value

[in] Defuzzification method type.

EmptyRule

Creates an empty fuzzy Mamdani rule based on the current system

```
CMamdaniFuzzyRule* EmptyRule()
```

Return Value

Fuzzy Mamdani rule.

ImplicationMethod

Sets a type of the conditions implication operator.

```
void ImplicationMethod(  
    ImplicationMethod value    // implication operator type
```



```
)
```

Parameters

value

[in] Type of the conditions implication operator.

Output

Gets the list of fuzzy Mamdani output variables.

```
CList* Output()
```

Return Value

List of fuzzy variables.

OutputByName

Gets a fuzzy Mamdani output variable by a specified name.

```
CFuzzyVariable* OutputByName(  
    const string name    // fuzzy variable name  
)
```

Parameters

name

[in] Fuzzy variable name.

Return Value

Fuzzy Mamdani variable with a specified name.

ParseRule

Creates a fuzzy Mamdani rule based on a specified line.

```
CMamdaniFuzzyRule* ParseRule(  
    const string rule    // string representation of a fuzzy rule  
)
```

Parameters

rule

[in] String representation of a fuzzy Mamdani rule.

Return Value

Fuzzy Mamdani rule.

Rules

Returns the list of fuzzy Mamdani rules.

```
CList* Rules ()
```

Return Value

List of fuzzy Mamdani rules.

Sugeno system

Sugeno fuzzy logic system is one of the two basic types of fuzzy systems. Output variable values are set as a linear combination of input variables.

Description

Unlike the Mamdani rule, an input variable value is set by a linear function from entries rather than by a fuzzy term. Fuzzy logic rule for the Sugeno algorithm can be described as follows:

if $(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n)$ *then* $(Y = b_0 + b_1 \cdot X_1 + b_2 \cdot X_2 + \dots + b_n \cdot X_n)(W)$

where:

- $X = (X_1, X_2, X_3 \dots X_n)$ – vector of input variables;
- Y – output variable;
- $a = (a_1, a_2, a_3 \dots a_n)$ – vector of input variable values;
- $b = (b_1, b_2, b_3 \dots b_n)$ – free term ratio in the linear function for an output value
- W – rule weight.

Class methods

Class method	Description
Calculate	Calculates a fuzzy inference for the system
CreateSugenoFunction	Creates a linear Sugeno function for the system.
EmptyRule	Creates an empty fuzzy Sugeno rule based on the current system.
Output	Gets the list of fuzzy Sugeno output variables.
OutputByName	Gets a fuzzy Sugeno output variable by a specified name.
ParseRule	Creates a fuzzy Sugeno rule based on a specified line.
Rules	Returns the list of fuzzy rules.

Methods inherited from class CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

Calculate

Calculates a fuzzy inference for the system

```
CList* Calculate(
    CList*& inputValues    // input data
)
```


Parameters*inputValues*

[in] Input data for calculation.

Return Value

Calculation result.

CreateSugenoFunction

Creates a linear Sugeno function for the system.

```

CLinearSugenoFunction* CreateSugenoFunction (
    const string  name,           // function name
    const double& coeffs[]        // function ratios
)

```

Parameters*name*

[in] Function name.

coeffs[]

[in] Function ratios.

Return Value

Linear Sugeno function.

Note

The size of the ratio array may be equal to a number of inputs or exceed that number by one. In the first case, the free term of the Sugeno linear function is equal to zero, while in the second case, it is equal to the last ratio.

CreateSugenoFunction

Creates a linear Sugeno function for the system.

```

CLinearSugenoFunction* CreateSugenoFunction (
    const string  name,           // function name
    CList*&       coeffs,         // list of pairs fuzzy variable - its ratio
    const double  constValue      // function free term ratio
)

```

Parameters*name*

[in] Function name.

coeffs[]

[in] Function ratios.

Return Value

Linear Sugeno function.

EmptyRule

Creates an empty fuzzy Sugeno rule based on the current system.

```
CSugenoFuzzyRule* EmptyRule()
```

Return Value

Fuzzy Sugeno rule.

Output

Gets the list of fuzzy Sugeno output variables.

```
CList* Output()
```

Return Value

List of fuzzy variables.

OutputByName

Gets a fuzzy Sugeno output variable by a specified name.

```
CSugenoVariable* OutputByName(  
    const string name    // fuzzy variable name  
)
```

Parameters

name

Fuzzy variable name.

Return Value

Fuzzy Sugeno variable with a specified name.

ParseRule

Creates a fuzzy Sugeno rule based on a specified line.

```
CSugenoFuzzyRule* ParseRule(  
    const string rule    // string representation of a fuzzy Sugeno rule  
)
```

Parameters

rule

[in] String representation of a fuzzy Sugeno rule.

Return Value

Fuzzy Sugeno rule.

Rules

Returns the list of fuzzy rules.

```
CList* Rules ()
```

Return Value

List of fuzzy rules.

Class for working with OpenCL programs

The COpenCL class is a wrapper to facilitate working with the [OpenCL functions](#). In some cases, use of the GPU allows to substantially increase the speed of computations.

Examples of class use for calculations based on float and double values can be found in the corresponding subdirectories of the MQL5\Scripts\Examples\OpenCL\ folder. The source codes of the OpenCL programs are located in MQL5\Scripts\Examples\OpenCL\Double\Kernels and MQL5\Scripts\Examples\OpenCL\Float\Kernels subdirectories.

- MatrixMult.mq5 - example of matrix multiplication using global and local memory
- BitonicSort.mq5 - example of parallel sorting of array elements in GPU
- FFT.mq5 - example of fast Fourier transform calculation
- Wavelet.mq5 - example of wavelet transform of data using the Morlet wavelet.

It is recommended to write the source code for OpenCL in separate CL files, which can later be included in the MQL5 program using the [resource variables](#).

Declaration

```
class COpenCL
```

Title

```
#include <OpenCL\OpenCL.mqh>
```

Class methods

Name	Description
BufferCreate	Creates an OpenCL buffer at the specified index
BufferFree	Deletes buffer at the specified index
BufferFromArray	Creates a buffer at the specified index from an array of values
BufferRead	Reads an OpenCL buffer at the specified index into an array
BufferWrite	Writes an array of values into buffer at the specified index
Execute	Executes the OpenCL kernel with the specified index
GetContext	Returns handle of the OpenCL context
GetKernel	Returns handle of the kernel object at the specified index
GetKernelName	Returns name of the kernel object at the specified index

<u>GetProgram</u>	Returns handle of the OpenCL program
<u>Initialize</u>	Initializes the OpenCL program
<u>KernelCreate</u>	Creates an entry point into the OpenCL program at the specified index
<u>KernelFree</u>	Removes an OpenCL start function at the specified index
<u>SetArgument</u>	Sets a parameter for the OpenCL function at the specified index
<u>SetArgumentBuffer</u>	Sets an OpenCL buffer as a parameter of the OpenCL function at the specified index
<u>SetArgumentLocalMemory</u>	Sets a parameter in local memory for the OpenCL function at the specified index
<u>SetBuffersCount</u>	Sets the number of buffers
<u>SetKernelsCount</u>	Sets the number of kernel objects
<u>Shutdown</u>	Unloads the OpenCL program
<u>SupportDouble</u>	Checks if floating point data types are supported on the device

BufferCreate

Creates an OpenGL buffer at the specified index.

```
bool BufferCreate(  
    const int   buffer_index,      // buffer index  
    const uint  size_in_bytes,     // buffer size in bytes  
    const uint  flags              // combination of flags that define the buffer properties  
);
```

Parameters

buffer_index

[in] Index buffer.

size_in_bytes

[in] Buffer size in bytes.

flags

[in] Buffer properties that are set using a combination of flags.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferFree

Deletes buffer at the specified index.

```
bool BufferFree(  
    const int  buffer_index    // buffer index  
);
```

Parameters

buffer_index
[in] Index buffer.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferFromArray

Creates a buffer at the specified index from an array of values.

```
template<typename T>
bool BufferFromArray(
    const int    buffer_index,           // buffer index
    T            &data[],               // array of values
    const uint   data_array_offset,     // offset in the array of values, in bytes
    const uint   data_array_count,     // number of values from the array to write
    const uint   flags                  // combination of flags that define the buffer p
);
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] An array of values to be written into the OpenCL buffer.

data_array_offset

[in] Offset in the array of values in bytes, from which writing of values begins.

data_array_count

[in] The number of values to be written.

flags

[in] Buffer properties that are set using a combination of flags.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferRead

Reads an OpenCL buffer at the specified index into an array.

```
template<typename T>
bool BufferRead(
    const int    buffer_index,           // buffer index
    T            &data[],               // array of values
    const uint   cl_buffer_offset,      // offset in the OpenCL buffer, in bytes
    const uint   data_array_offset,     // shift in the array elements
    const uint   data_array_count       // number of values from the buffer to read
);
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] Array to obtain the values of the OpenCL buffer.

cl_buffer_offset

[in] Offset in the OpenCL buffer in bytes, from which to start reading values.

data_array_offset

[in] Index of the first element of the array to write values of the OpenCL buffer.

data_array_count

[in] The number of values to be read.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferWrite

Writes an array of values into buffer at the specified index.

```
template<typename T>
bool BufferWrite(
    const int    buffer_index,           // buffer index
    T            &data[],               // array of values
    const uint   cl_buffer_offset,      // offset in the OpenCL buffer, in bytes
    const uint   data_array_offset,     // shift in the array elements
    const uint   data_array_count       // number of values from the array to write
);
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] An array of values to be written into the OpenCL buffer.

cl_buffer_offset

[in] Offset in the OpenCL buffer in bytes, from which to start writing values.

data_array_offset

[in] Index of the first element of the array, starting from which the array values are written to the OpenCL buffer.

data_array_count

[in] The number of values to be written.

Return Value

In case of successful execution, returns true, otherwise - false.

Execute

Executes the OpenCL program at the specified index.

```
bool Execute(  
    const int    kernel_index,           // index of the kernel  
    const int    work_dim,              // dimension of the tasks space  
    const uint   &work_offset[],        // initial offset in the tasks space  
    const uint   &work_size[]           // total number of tasks  
);
```

Executes the OpenCL kernel with the specified index and number of tasks in the local group.

```
bool Execute(  
    const int    kernel_index,           // index of the kernel  
    const int    work_dim,              // dimension of the tasks space  
    const uint   &work_offset[],        // initial offset in the tasks space  
    const uint   &work_size[],          // total number of tasks  
    const uint   &local_work_size[]     // number of tasks in the local group  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

work_dim

[in] Dimension of the tasks space.

&work_offset[]

[in][out] Initial offset in the tasks space. Passed by reference.

&work_size[]

[in][out] The size of the tasks subset. Passed by reference.

&local_work_size[]

[in][out] The size of the local tasks subset in the group. Passed by reference.

Return Value

In case of successful execution, returns true, otherwise - false.

GetContext

Returns handle of the OpenCL context.

```
int GetContext();
```

Return Value

Handle of the OpenCL context.

GetKernel

Returns handle of the kernel object at the specified index.

```
int GetKernel(  
    const int kernel_index    // index of the kernel  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

Handle of the kernel object.

GetKernelName

Returns name of the kernel object at the specified index.

```
string GetKernelName(  
    const int kernel_index    // index of the kernel  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

Name of the kernel object.

GetProgram

Returns handle of the OpenCL program.

```
int GetProgram();
```

Return Value

Handle of the OpenCL program.

Initialize

Initializes the OpenCL program.

```
bool Initialize(  
    const string program,           // handle of the OpenCL program  
    const bool show_log=true       // keep a log  
);
```

Parameters

program

[in] Handle of the OpenCL program.

show_log=true

[in] Enable logging messages.

Return Value

Returns true, if the initialization succeeded. Otherwise, it returns false.

KernelCreate

Creates an entry point into the OpenCL program at the specified index.

```
bool KernelCreate(  
    const int    kernel_index,    // index of the kernel  
    const string kernel_name      // name of the kernel  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

kernel_name

[in] Name of the kernel object.

Return Value

In case of successful execution, returns true, otherwise - false.

KernelFree

Removes an OpenCL start function at the specified index.

```
bool KernelFree(  
    const int kernel_index    // index of the kernel  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgument

Sets a parameter for the OpenCL function at the specified index.

```
template<typename T>
bool SetArgument (
    const int kernel_index,    // index of the kernel
    const int arg_index,      // index of the function argument
    T value                    // source code
);
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

value

[in] The value of the function argument.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgumentBuffer

Sets an OpenCL buffer as a parameter of the OpenCL function at the specified index.

```
bool SetArgumentBuffer(  
    const int kernel_index,    // index of the kernel  
    const int arg_index,      // index of the function argument  
    const int buffer_index    // buffer index  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

buffer_index

[in] Index buffer.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgumentLocalMemory

Sets a parameter in local memory for the OpenCL function at the specified index.

```
bool SetArgumentLocalMemory(  
    const int kernel_index,           // index of the kernel  
    const int arg_index,             // index of the function argument  
    const int local_memory_size      // size of the local memory  
);
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

local_memory_size

[in] Size of the local memory.

Return Value

In case of successful execution, returns true, otherwise - false.

SetBuffersCount

Sets the number of buffers.

```
bool SetBuffersCount(  
    const int total_buffers    // number of buffers  
);
```

Parameters

total_buffers

[in] The total number of buffers.

Return Value

In case of successful execution, returns true, otherwise - false.

SetKernelsCount

Sets the number of kernel objects.

```
bool SetKernelsCount(  
    const int total_kernels    // number of kernels  
);
```

Parameters

total_kernels

[in] The total number of kernels.

Return Value

In case of successful execution, returns true, otherwise - false.

Shutdown

Unloads the OpenCL program.

```
void Shutdown();
```

Return Value

No return value.

SupportDouble

Checks if floating point data types are supported on the device.

```
bool SupportDouble();
```

Return Value

Returns true, if the device supports floating point data types.

Basic Class CObject

Class CObject is the base class for constructing a MQL5 Standard Library.

Description

Class CObject allows all its descendants to be part of a linked list. Also, a number of virtual methods for further implementation in descendant classes are identified.

Declaration

```
class CObject
```

Title

```
#include <Object.mqh>
```

Inheritance hierarchy

CObject

Direct descendants

[CAccountInfo](#), [CArray](#), [CChart](#), [CChartObject](#), [CCurve](#), [CDealInfo](#), [CDictionary_Obj_Double](#), [CDictionary_Obj_Obj](#), [CDictionary_String_Obj](#), [CExpertBase](#), [CFile](#), [CHistoryOrderInfo](#), [CList](#), [COrderInfo](#), [CPositionInfo](#), [CString](#), [CSymbolInfo](#), [CTerminalInfo](#), [CTrade](#), [CTreeNode](#), [CWnd](#), [ICondition](#), [IExpression](#), [IMembershipFunction](#), [INamedValue](#), [IParsableRule](#)

Class Methods by Groups

Attributes	
Prev	Gets the value of the previous item
Prev	Sets the value of the previous item
Next	Gets the value of the subsequent element
Next	Sets the next element
Compare methods	
virtual Compare	Returns the result of comparison with another object
Input/output	
virtual Save	Writes object to a file
virtual Load	Reads the object from the file
virtual Type	Returns the type of object

Prev

Gets a pointer to the previous element in the list.

```
CObject* Prev()
```

Return Value

Pointer to the previous element in the list. If an item is listed first, then return NULL.

Example:

```
//--- example for CObject::Prev()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
    //---
    object_first=new CObject;
    if(object_first==NULL)
    {
        printf("Object create error");
        return;
    }
    object_second=new CObject;
    if(object_second==NULL)
    {
        printf("Object create error");
        delete object_first;
        return;
    }
    //--- set interconnect
    object_first.Next(object_second);
    object_second.Prev(object_first);
    //--- use prev object
    CObject *object=object_second.Prev();
    //--- delete objects
    delete object_first;
    delete object_second;
}
```


Prev

Sets the pointer to the previous element in the list.

```
void Prev(  
    CObject* object    // Pointer to the previous element in the list  
)
```

Parameters

object

[in] New value of the pointer to the previous element in the list.

Example:

```
///--- example for CObject::Prev(CObject*)  
#include <Object.mqh>  
///---  
void OnStart()  
{  
    CObject *object_first,*object_second;  
    ///---  
    object_first=new CObject;  
    if(object_first==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    object_second=new CObject;  
    if(object_second==NULL)  
    {  
        printf("Object create error");  
        delete object_first;  
        return;  
    }  
    ///--- set interconnect  
    object_first.Next(object_second);  
    object_second.Prev(object_first);  
    ///--- use objects  
    ///--- ...  
    ///--- delete objects  
    delete object_first;  
    delete object_second;  
}
```


Next

Gets a pointer to the next element in the list.

```
CObject* Next()
```

Return Value

Pointer to the next element in the list. If this is the last element in the list, return NULL.

Example:

```
//--- example for CObject::Next()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
    //---
    object_first=new CObject;
    if(object_first==NULL)
    {
        printf("Object create error");
        return;
    }
    object_second=new CObject;
    if(object_second==NULL)
    {
        printf("Object create error");
        delete object_first;
        return;
    }
    //--- set interconnect
    object_first.Next(object_second);
    object_second.Prev(object_first);
    //--- use next object
    CObject *object=object_first.Next();
    //--- delete objects
    delete object_first;
    delete object_second;
}
```


Next

Sets the pointer to the next element in the list.

```
void Next(  
    CObject* object    // Pointer to the next element in the list  
)
```

Parameters

object

[in] New value of the pointer to the next element in the list.

Example:

```
//--- example for CObject::Next(CObject*)  
#include <Object.mqh>  
//---  
void OnStart()  
{  
    CObject *object_first,*object_second;  
    //---  
    object_first=new CObject;  
    if(object_first==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    object_second=new CObject;  
    if(object_second==NULL)  
    {  
        printf("Object create error");  
        delete object_first;  
        return;  
    }  
    //--- set interconnect  
    object_first.Next(object_second);  
    object_second.Prev(object_first);  
    //--- use objects  
    //--- ...  
    //--- delete objects  
    delete object_first;  
    delete object_second;  
}
```


Compare

Compares the data on a list element with data on another list element.

```
virtual int Compare(  
    const CObject* node,      // element  
    const int      mode=0     // variant  
) const
```

Parameters

node

[in] Pointer to a list element to compare

mode=0

[in] Comparison variant

Return Value

0 - in case the list elements are equal, -1 - if the list element is less than the element used for comparison (node), 1 - if the list element is greater than the element used for comparison (node).

Note

Compare() method in CObject class always returns 0 and does not perform any action. If you want to compare data in derived class, the Compare(...) method should be implemented. The 'mode' parameter should be used when implementing multivariate comparison.

Example:

```
//--- example for CObject::Compare(...)  
#include <Object.mqh>  
//---  
void OnStart()  
{  
    CObject *object_first,*object_second;  
    //---  
    object_first=new CObject;  
    if(object_first==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    object_second=new CObject;  
    if(object_second==NULL)  
    {  
        printf("Object create error");  
        delete object_first;  
        return;  
    }  
    //--- set interconnect  
    object_first.Next(object_second);
```



```
object_second.Prev(object_first);  
//--- compare objects  
int result=object_first.Compare(object_second);  
//--- delete objects  
delete object_first;  
delete object_second;  
}
```


Save

Saves list element data in a file.

```
virtual bool Save(  
    int file_handle    // File handle  
)
```

Parameters

file_handle

[in] Handle of the binary file opened earlier using the FileOpen () function

Return Value

true - successfully completed, false - error.

Note

Save(int) method in CObject class always returns 'true' and does not perform any action. If you want to save the data of a derived class in a file, the Save(int) method should be implemented.

Example:

```
//--- example for CObject::Save(int)  
#include <Object.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CObject *object=new CObject;  
    //---  
    if(object!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- set objects data  
    //--- . . .  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!object.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete object;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
    }  
}
```



```
        FileClose(file_handle);  
    }  
    delete object;  
}
```


Load

Loads list element data from a file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file opened earlier using the FileOpen() function.

Return Value

true - successfully completed, false - error.

Note

Load(int) method in the CObject class always returns 'true' and does not perform any action. If you want to load the data of a derived class from a file, the Load(int) method should be implemented.

Example:

```
//--- example for CObject::Load(int)  
#include <Object.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CObject *object=new CObject;  
    //---  
    if(object!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!object.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete object;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
}
```



```
//--- use object  
//--- . . .  
delete object;  
}
```


Type

Gets the type identifier.

```
virtual int Type() const
```

Return Value

Type identifier (for CObject - 0).

Example:

```
//--- example for CObject::Type()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object=new CObject;
    //---
    object=new CObject;
    if(object ==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get objects type
    int type=object.Type();
    //--- delete object
    delete object;
}
```


Data Structures

This section contains the technical details on working with various data structures (arrays, linked lists, etc.) and description of the relevant components of the MQL5 Standard Library.

Using classes of data structures will save time when creating custom data stores of various formats (including composite data structures).

MQL5 Standard Library (in terms of data sets) is placed in the working directory of the terminal in the Include\Arrays folder.

Data Arrays

Use of classes of dynamic data arrays will save time when creating a custom data stores of various formats (including multidimensional arrays).

MQL5 Standard Library (in terms of arrays of data) is located in the working directory of the terminal in the Include\Arrays folder.

Class	Description
<u>CArray</u>	Base class of dynamic data array
<u>CArrayChar</u>	Dynamic array of char or uchar variables
<u>CArrayShort</u>	Dynamic array of short or ushort variables
<u>CArrayInt</u>	Dynamic array of int or uint variables
<u>CArrayLong</u>	Dynamic array of long or ulong variables
<u>CArrayFloat</u>	Dynamic array of float variables
<u>CArrayDouble</u>	Dynamic array of double variables
<u>CArrayString</u>	Dynamic array of string variables
<u>CArrayObj</u>	Dynamic array of <u>CObject</u> pointers
<u>CList</u>	Provides the ability to work with a list of instances of <u>CObject</u> class and its descendants
<u>CTreeNode</u>	Provides the ability to work with nodes of the <u>CTree</u> binary tree
<u>CTree</u>	Provides the ability to work with the binary tree of the <u>CTreeNode</u> class instances and its descendants

CArray

CArray class is the base class of a dynamic array of variables.

Description

Class CArray is intended to operate on dynamic arrays of variables: memory allocation, sorting, and working with files.

Declaration

```
class CArray : public CObject
```

Title

```
#include <Arrays\Array.mqh>
```

Inheritance hierarchy

CObject

CArray

Direct descendants

CArrayChar, CArrayDouble, CArrayFloat, CArrayInt, CArrayLong, CArrayObj, CArrayShort, CArrayString

Class Methods by Groups

Attributes	
<u>Step</u>	Gets the increment size of the array
<u>Step</u>	Sets the increment size of the array
<u>Total</u>	Gets the number of elements in the array
<u>Available</u>	Gets the number of free elements of the array that are available without additional memory allocation
<u>Max</u>	Gets the maximum possible size of the array without memory reallocation
<u>IsSorted</u>	Gets the flag of array being sorted using specified sorting mode
<u>SortMode</u>	Gets the sorting mode for an array
Clear methods	
<u>Clear</u>	Deletes all of the array elements without memory release
Sort methods	

Sort	Sorts an array to the specified option
Input/output	
virtual Save	Saves data array in a file
virtual Load	Loads data array from a file

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Step

Gets the increment size of the array.

```
int Step() const
```

Return Value

Increment size of the array.

Example:

```
//--- example for CArray::Step()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get resize step
    int step=array.Step();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```


Step

Sets the increment size of the array.

```
bool Step(  
    int step    // step  
)
```

Parameters

step

[in] The new value of the increment size of the array.

Return Value

true - successful, false - there was an attempt to establish a step less than or equal to zero.

Example:

```
//--- example for CArray::Step(int)  
#include <Arrays\Array.mqh>  
//---  
void OnStart()  
{  
    CArray *array=new CArray;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- set resize step  
    bool result=array.Step(1024);  
    //--- use array  
    //--- ...  
    //--- delete array  
    delete array;  
}
```


Total

Gets the number of elements in the array.

```
int Total() const;
```

Return Value

Number of elements in the array.

Example:

```
//--- example for CArray::Total()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check total
    int total=array.Total();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```


Available

Gets the number of free elements of the array that are available without additional memory allocation.

```
int Available() const
```

Return Value

Number of free elements of the array available without additional memory allocation.

Example:

```
//--- example for CArray::Available()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check available
    int available=array.Available();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```


Max

Gets the maximum possible size of the array without memory reallocation.

```
int Max() const
```

Return Value

The maximum possible size of the array without memory reallocation.

Example:

```
//--- example for CArray::Max()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check maximum size
    int max=array.Max();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```


IsSorted

Gets the flag of array being sorted using the specified sorting mode.

```
bool IsSorted(  
    int mode=0      // sorting mode  
) const
```

Parameters

mode=0

[in] Tested sorting mode.

Return Value

Flag of the sorted list. If the list is sorted using the specified mode - true, otherwise - false.

Note

The sort flag cannot be changed directly. It is set by the Sort() method and reset by any add/insert method except for the InsertSort(...).

Example:

```
///--- example for CArray::IsSorted()  
#include <Arrays\Array.mqh>  
///---  
void OnStart()  
{  
    CArray *array=new CArray;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- check sorted  
    if(array.IsSorted())  
    {  
        ///--- use methods for sorted array  
        ///--- ...  
    }  
    ///--- delete array  
    delete array;  
}
```


SortMode

Gets the sorting mode for an array.

```
int SortMode() const;
```

Return Value

Sorting mode.

Example:

```
//--- example for CArray::SortMode()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sort mode
    int sort_mode=array.SortMode();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```


Clear

Deletes all of the array elements without memory release.

```
void Clear()
```

Return Value

None.

Example:

```
//--- example for CArray::Clear()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- use array
    //--- ...
    //--- clear array
    array.Clear();
    //--- delete array
    delete array;
}
```


Sort

Sorts an array using the specified option.

```
void Sort(  
    int mode=0      // sorting mode  
)
```

Parameters

mode=0

[in] Mode of array sorting.

Return Value

No.

Note

Sorting an array is always ascending. For arrays of primitive data types (CArrayChar, CArrayShort, etc.), the 'mode' parameter is not used. For the CArrayObj array, multivariate sorting should be implemented in the Sort(int) method of derived classes.

Example:

```
//--- example for CArray::Sort(int)  
#include <Arrays\Array.mqh>  
//---  
void OnStart()  
{  
    CArray *array=new CArray;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- sorting by mode 0  
    array.Sort(0);  
    //--- use array  
    //--- ...  
    //--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of a binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArray::Save(int)  
#include <Arrays\Array.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArray *array=new CArray;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- delete array  
    delete array;  
}
```


Load

Loads data array from a file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of a binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArray::Load(...)  
#include <Arrays\Array.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArray *array=new CArray;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- delete array  
    delete array;  
}
```


CArrayChar

CArrayChar is a class of dynamic array of char or uchar variables.

Description

CArrayChar class provides the ability to work with a dynamic array of char or uchar variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayChar : public CArray
```

Title

```
#include <Arrays\ArrayChar.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayChar

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

AssignArray	Copies the elements of one array to another
Modify methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array methods	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array

Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the array type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is increased by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayChar::Reserve(int)
#include <Arrays\ArrayChar.mqh>
///---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size    // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayChar::Resize(int)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayChar::Shutdown()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    char element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayChar::Add(char)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const char& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayChar::AddArray(const char &[])  
#include <Arrays\ArrayChar.mqh>  
///---  
char src[];  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayChar* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayChar class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayChar::AddArray(const CArrayChar*)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayChar *src=new CArrayChar;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    char element,    // element to insert  
    int pos          // position  
)
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayChar::Insert(char,int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(  
    const char&  src[],      // source array  
    int          pos        // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayChar::InsertArray(const char &[],int)  
#include <Arrays\ArrayChar.mqh>  
//---  
char src[];  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayChar* src,      // pointer to the source  
    int pos              // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayChar::InsertArray(const CArrayChar*,int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayChar *src=new CArrayChar;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //--- delete source array  
    delete src;  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const char& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayChar::AssignArray(const char &[])  
#include <Arrays\ArrayChar.mqh>  
///---  
char src[];  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayChar* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
///--- example for CArrayChar::AssignArray(const CArrayChar*)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayChar *src =new CArrayChar;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- arrays are identical  
    ///--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,           // position  
    char   element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayChar::Update(int,char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,'A'))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // value  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayChar::Shift(int,int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayChar::Delete(int)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of the last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayChar::DeleteRange(int,int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
char At(  
    int pos    // position  
    ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, CHAR_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, CHAR_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayChar::At(int)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        char result=array.At(i);  
        if(result==CHAR_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- error of reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const char& src[]      // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayChar::CompareArray(const char &[])  
#include <Arrays\ArrayChar.mqh>  
//---  
char src[];  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete array  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const CArrayChar* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayChar::CompareArray(const CArrayChar*)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayChar *src=new CArrayChar;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    char element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayChar::InsertSort(char)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort('A'))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::Search(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search('A')!=-1) printf("Element found");  
    else                      printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchGreat(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat('A')!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLess(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLess('A')!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    char element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayChar::SearchGreatOrEqual(char)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual('A')!=-1) printf("Element found");  
    else                                printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLessOrEqual(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual('A')!=-1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchFirst(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst('A')!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLast(char)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLast('A')!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    char element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayChar::SearchLinear(char)  
#include <Arrays\ArrayChar.mqh>  
///---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear('A')!=-1) printf("Element found");  
    else                          printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayChar::Save(int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- delete array  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayChar::Load(int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = '%c'",i,array.At(i));  
    }  
}
```



```
    }  
    ///--- delete array  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayChar - 77).

Example:

```
//--- example for CArrayChar::Type()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayShort

CArrayShort is a class of dynamic array of short or ushort variables.

Description

Class CArrayShort provides the ability to work with a dynamic array of short or ushort variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayShort : public CArray
```

Title

```
#include <Arrays\ArrayShort.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayShort

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array

Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayShort::Reserve(int)
#include <Arrays\ArrayShort.mqh>
///---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size      // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayShort::Resize(int)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayShort::Shutdown()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    short element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayShort::Add(short)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const short& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayShort::AddArray(const short &[])  
#include <Arrays\ArrayShort.mqh>  
///---  
short src[];  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayShort* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayShort class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayShort::AddArray(const CArrayShort*)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayShort *src=new CArrayShort;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    short element,    // element to insert  
    int pos           // position  
)
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayShort::Insert(short,int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(  
    const short&  src[],      // source array  
    int           pos         // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayShort::InsertArray(const short &[],int)  
#include <Arrays\ArrayShort.mqh>  
//---  
short src[];  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayShort* src,      // pointer to the source  
    int pos               // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements to insert.

pos

[in] Position in the array to insert.

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayShort::InsertArray(const CArrayShort*,int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayShort *src=new CArrayShort;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //--- delete source array  
    delete src;  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const short& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayShort::AssignArray(const short &[])  
#include <Arrays\ArrayShort.mqh>  
///---  
short src[];  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayShort* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayShort::AssignArray(const CArrayShort*)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayShort *src =new CArrayShort;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    //--- arrays is identical  
    //--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,          // position  
    short  element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayShort::Update(int,short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,100))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // positions  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayShort::Shift(int,int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayShort::Delete(int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of the last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayShort::DeleteRange(int,int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
short At(  
    int pos    // position  
    ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, SHORT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, SHORT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayShort::At(int)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        short result=array.At(i);  
        if(result==SHORT_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- error of reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const short& src[]      // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayShort::CompareArray(const short &[])  
#include <Arrays\ArrayShort.mqh>  
//---  
short src[];  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete array  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const CArrayShort* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayShort::CompareArray(const CArrayShort*)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayShort *src=new CArrayShort;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    short element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayShort::InsertSort(short)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(100))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::Search(short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search(100)!=-1) printf("Element found");  
    else                     printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - element not found.

Example:

```
//--- example for CArrayShort::SearchGreat(short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat(100)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayShort::SearchLess(short)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLess(100)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayShort::SearchGreatOrEqual(short)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual(100)!=-1) printf("Element found");  
    else                                printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchLessOrEqual(short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual(100) != -1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchFirst(short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst(100)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchLast(short)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLast(100)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    short element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayShort::SearchLinear(short)  
#include <Arrays\ArrayShort.mqh>  
///---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear(100) != -1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayShort::Save(int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(i);  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayShort::Load(int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = %d",i,array.At(i));  
    }  
}
```



```
    }  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayShort - 82).

Example:

```
//--- example for CArrayShort::Type()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayInt

CArrayInt is a class of dynamic array of int or uint variables.

Description

The class CArrayInt provides the ability to work with a dynamic array of int or uint variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayInt : public CArray
```

Title

```
#include <Arrays\ArrayInt.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayInt

Direct descendants

[CSpreadBuffer](#)

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position

InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array

SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayInt::Reserve(int)
#include <Arrays\ArrayInt.mqh>
///---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size      // number  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayInt::Resize(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayInt::Shutdown()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    int element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayInt::Add(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const int& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayInt::AddArray(const int &[])  
#include <Arrays\ArrayInt.mqh>  
//---  
int src[];  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayInt* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayInt class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayInt::AddArray(const CArrayInt*)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayInt *src=new CArrayInt;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    //--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    int element,    // element to insert  
    int pos         // position  
)
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayInt::Insert(int,int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(  
    const int&  src[],      // source array  
    int         pos        // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayInt::InsertArray(const int &[],int)  
#include <Arrays\ArrayInt.mqh>  
//---  
int src[];  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayInt* src,      // pointer to the source  
    int pos             // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements to insert.

pos

[in] Position in the array to insert.

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayInt::InsertArray(const CArrayInt*,int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayInt *src=new CArrayInt;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //-- delete source array  
    delete src;  
    //-- use array  
    //-- . . .  
    //-- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const int& src[]    // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayInt::AssignArray(const int &[])  
#include <Arrays\ArrayInt.mqh>  
///---  
int src[];  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayInt* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
///--- example for CArrayInt::AssignArray(const CArrayInt*)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayInt *src =new CArrayInt;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
}
```



```
//--- arrays is identical
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int pos,           // position  
    int element        // value  
)
```

Parameters

pos

[in] Position of the element in the array to change.

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayInt::Update(int,int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,10000))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayInt::Shift(int,int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayInt::Delete(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of the last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayInt::DeleteRange(int,int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
int At(  
    int pos    // position  
) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, INT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, INT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayInt::At(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        int result=array.At(i);  
        if(result==INT_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- error of reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const int& src[]    // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayInt::CompareArray(const int &[])  
#include <Arrays\ArrayInt.mqh>  
///---  
int src[];  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete array  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const CArrayInt* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayInt::CompareArray(const CArrayInt*)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayInt *src=new CArrayInt;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    int element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayInt::InsertSort(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(10000))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::Search(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search(10000) != -1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchGreat(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat(10000) != -1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLess(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLess(10000) != -1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    int element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayInt::SearchGreatOrEqual(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual(10000)!=-1) printf("Element found");  
    else                                  printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    int element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLessOrEqual(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual(10000)!=-1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt:: SearchFirst(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst(10000)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLast(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLast(10000) != -1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    int element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayInt::SearchLinear(int)  
#include <Arrays\ArrayInt.mqh>  
///---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear(10000) != -1) printf("Element found");  
    else                               printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayInt::Save(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(i);  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayInt::Load(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = %d",i,array.At(i));  
    }  
}
```



```
    }  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayInt - 82).

Example:

```
//--- example for CArrayInt::Type()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayLong

CArrayLong class is a class of dynamic array of long or ulong variables.

Description

The CArrayLong class provides the ability to work with a dynamic array of long or ulong variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayLong : public CArray
```

Title

```
#include <Arrays\ArrayLong.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayLong

Direct descendants

[CRealVolumeBuffer](#), [CTickVolumeBuffer](#), [CTimeBuffer](#)

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position

InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array

SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayLong::Reserve(int)
#include <Arrays\ArrayLong.mqh>
///---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size    // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayLong::Resize(int)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayLong::Shutdown()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    long element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayLong::Add(long)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const long& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayLong::AddArray(const long &[])  
#include <Arrays\ArrayLong.mqh>  
//---  
long src[];  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayLong* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayLong class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayLong::AddArray(const CArrayLong*)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayLong *src=new CArrayLong;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    long element,    // element to insert  
    int pos          // position  
)
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayLong::Insert(long,int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(  
    const long&  src[],      // source array  
    int          pos         // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayLong::InsertArray(const long &[],int)  
#include <Arrays\ArrayLong.mqh>  
//---  
long src[];  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayLong* src,      // pointer to the source  
    int pos              // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayLong::InsertArray(const CArrayLong*,int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayLong *src=new CArrayLong;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //-- delete source array  
    delete src;  
    //-- use array  
    //-- . . .  
    //-- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const long& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayLong::AssignArray(const long &[])  
#include <Arrays\ArrayLong.mqh>  
///---  
long src[];  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayLong* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
///--- example for CArrayLong::AssignArray(const CArrayLong*)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayLong *src =new CArrayLong;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- arrays is identical  
    ///--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,          // position  
    long   element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayLong::Update(int,long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,1000000))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayLong::Shift(int,int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayLong::Delete(int)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of the last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayLong::DeleteRange(int,int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
long At(  
    int pos    // position  
    ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, LONG_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, LONG_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayLong::At(int)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        long result=array.At(i);  
        if(result==LONG_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- Error reading from the array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const long& src[]      // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayLong::CompareArray(const long &[])  
#include <Arrays\ArrayLong.mqh>  
//---  
long src[];  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete array  
    delete array;  
}
```


CompareArrayconst

Compares the array with another one.

```
bool CompareArrayconst(  
    const CArrayLong* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayLong::CompareArray(const CArrayLong*)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayLong *src=new CArrayLong;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    long element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayLong::InsertSort(long)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(1000000))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::Search(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search(1000000)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchGreat(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat(1000000) != -1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLess(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLess(1000000)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    long element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchGreatOrEqual(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreatOrEqual(1000000)!=-1) printf("Element found");  
    else                                     printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLessOrEqual(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual(1000000) != -1) printf("Element found");  
    else                                     printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchFirst(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst(1000000)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLast(long)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLast(1000000)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    long element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayLong::SearchLinear(long)  
#include <Arrays\ArrayLong.mqh>  
///---  
void OnStart()  
{  
    CArrayLong *array=new CArrayLong;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear(1000000) != -1) printf("Element found");  
    else                                printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayLong::Save(int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(i);  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayLong::Load(int)  
#include <Arrays\ArrayLong.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayLong *array=new CArrayLong;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = %I64",i,array.At(i));  
    }  
}
```



```
    }  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayLong - 84).

Example:

```
//--- example for CArrayLong::Type()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayFloat

CArrayFloat class is a class of dynamic array of float variables.

Description

The CArrayFloat class provides the ability to work with a dynamic array of float variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayFloat : public CArray
```

Title

```
#include <Arrays\ArrayFloat.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayFloat

Class Methods by Groups

Attributes	
Delta	Sets the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position

InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array

SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Delta

Sets the comparison tolerance.

```
void Delta(  
    float delta    // tolerance  
)
```

Parameters

delta

[in] The new value of the comparison tolerance.

Return Value

None

Note

Comparison tolerance is used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
///--- example for CArrayFloat::Delta(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- set compare variation  
    array.Delta(0.001);  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayFloat::Reserve(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool  Resize(  
    int  size      // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayFloat::Resize(int)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayFloat::Shutdown()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    float element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayFloat::Add(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const float& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayFloat::AddArray(const float &[])  
#include <Arrays\ArrayFloat.mqh>  
///---  
float src[];  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayFloat* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayFloat class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayFloat::AddArray(const CArrayFloat*)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayFloat *src=new CArrayFloat;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    float element,    // element to insert  
    int pos           // position  
)
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayFloat::Insert(float,int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    const float& src[],    // source array  
    int pos               // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayFloat::InsertArray(const float &[],int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
float src[];  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayFloat* src,      // pointer to the source  
    int pos               // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayFloat::InsertArray(const CArrayFloat*,int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayFloat *src=new CArrayFloat;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //--- delete source array  
    delete src;  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const float& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayFloat::AssignArray(const float &[])  
#include <Arrays\ArrayFloat.mqh>  
///---  
float src[];  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayFloat* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
///--- example for CArrayFloat::AssignArray(const CArrayFloat*)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayFloat *src =new CArrayFloat;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- arrays is identical  
    ///--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,          // position  
    float  element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayFloat::Update(int,float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,100.0))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayFloat::Shift(int,int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayFloat::Delete(int)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayFloat::DeleteRange(int,int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
float At(  
    int pos    // position  
) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, FLT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, FLT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayFloat::At(int)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        float result=array.At(i);  
        if(result==FLT_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- error reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const float& src[]    // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayFloat::CompareArray(const float &[])  
#include <Arrays\ArrayFloat.mqh>  
//---  
float src[];  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete array  
    delete array;  
}
```


AssignArrayconst

Compares the array with another one.

```
bool AssignArrayconst(  
    const CArrayFloat* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayFloat::CompareArray(const CArrayFloat*)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayFloat *src=new CArrayFloat;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    float element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayFloat::InsertSort(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(100.0))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    float element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::Search(float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search(100.0)!=-1) printf("Element found");  
    else                        printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    float element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchGreat(float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat(100.0)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    float element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat:: SearchLess(float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLess(100.0)!=-1) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    float element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayFloat::SearchGreatOrEqual(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");  
    else                                  printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    float element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchLessOrEqual(float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    float element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchFirst(float)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst(100.0)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    float element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayFloat::SearchLast(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLast(100.0)!=-1) printf("Element found");  
    else                           printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    float element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayFloat::SearchLinear(float)  
#include <Arrays\ArrayFloat.mqh>  
///---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear(100.0) != -1) printf("Element found");  
    else                               printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayFloat::Save(int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(i);  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayFloat::Load(int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = %f",i,array.At(i));  
    }  
}
```



```
    }  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayFloat - 87).

Example:

```
//--- example for CArrayFloat::Type()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayDouble

CArrayDouble class is a class of dynamic array of double variables.

Description

The CArrayDouble class provides the ability to work with a dynamic array of double variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayDouble : public CArray
```

Title

```
#include <Arrays\ArrayDouble.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayDouble

Direct descendants

[CDoubleBuffer](#)

Class Methods by Groups

Attributes	
Delta	Set the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array

InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Search for min/max	
Minimum	Gets the lowest value index in the specified range
Maximum	Gets the highest value index in the specified range
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array

<u>SearchGreatOrEqual</u>	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
<u>SearchLessOrEqual</u>	Searches for an element with a value less than or equal to the value of the sample in the sorted array
<u>SearchFirst</u>	Searches for the first element equal to the sample in the sorted array
<u>SearchLast</u>	Searches for the last element equal to the sample in the sorted array
<u>SearchLinear</u>	Searches for the element equal to the sample in the array
Input/output	
virtual <u>Save</u>	Saves data array in the file
virtual <u>Load</u>	Loads data array from the file
virtual <u>Type</u>	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Delta

Sets the comparison tolerance.

```
void Delta(  
    double delta    // tolerance  
)
```

Parameters

delta

[in] The new value of the comparison tolerance.

Return Value

No

Note

Comparison tolerance is used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
///--- example for CArrayDouble::Delta(double)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- set compare variation  
    array.Delta(0.001);  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayDouble::Reserve(int)
#include <Arrays\ArrayDouble.mqh>
///---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size      // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayDouble::Resize(int)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayDouble::Shutdown()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    double element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayDouble::Add(double)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(i))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const double& src[]    // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayDouble::AddArray(const double &[])  
#include <Arrays\ArrayDouble.mqh>  
///---  
double src[];  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayDouble* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayDouble class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayDouble::AddArray(const CArrayDouble*)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayDouble *src=new CArrayDouble;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    double element,    // element to insert  
    int pos            // position  
)
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayDouble::Insert(double,int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(i,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    const double& src[],      // source array  
    int pos                  // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayDouble::InsertArray(const double &[],int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
double src[];  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayDouble* src,      // pointer to the source  
    int pos                // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayDouble::InsertArray(const CArrayDouble*,int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayDouble *src=new CArrayDouble;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //-- delete source array  
    delete src;  
    //-- use array  
    //-- . . .  
    //-- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const double& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayDouble::AssignArray(const double &[])  
#include <Arrays\ArrayDouble.mqh>  
///---  
double src[];  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayDouble* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
///--- example for CArrayDouble::AssignArray(const CArrayDouble*)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayDouble *src =new CArrayDouble;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- arrays is identical  
    ///--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,           // position  
    double element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element.

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayDouble::Update(int,double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,100.0))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayDouble::Shift(int,int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayDouble::Delete(int)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of the last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayDouble::DeleteRange(int,int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
double At(  
    int pos      // position  
) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, DBL_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, DBL_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayDouble::At(int)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        double result=array.At(i);  
        if(result==DBL_MAX && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- Error reading from the array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const double& src[]      // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayDouble::CompareArray(const double &[])  
#include <Arrays\ArrayDouble.mqh>  
///---  
double src[];  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete array  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const CArrayDouble* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayDouble::CompareArray(const CArrayDouble*)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayDouble *src=new CArrayDouble;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- compare with another array  
    int result=array.CompareArray(src);  
    //--- delete arrays  
    delete src;  
    delete array;  
}
```


Minimum

Gets index of the lowest element of the array in the specified range.

```
int Minimum(  
    int start,      // starting index  
    int count       // number of elements  
) const
```

Parameters

start

[in] Starting index of the search range.

count

[in] Search range size (number of elements).

Return Value

Index of the lowest element in the specified range.

Maximum

Gets index of the highest element of the array in the specified range.

```
int Maximum(  
    int start,      // starting index  
    int count       // number of elements  
) const
```

Parameters

start

[in] Starting index of the search range.

count

[in] Search range size (number of elements).

Return Value

Index of the highest element in the specified range.

InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    double element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayDouble::InsertSort(double)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(100.0))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::Search(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search(100.0)!=-1) printf("Element found");  
    else                        printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchGreat(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat(100.0)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble:: SearchLess(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLess(100.0)!=-1) printf("Element found");  
    else                           printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    double element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayDouble::SearchGreatOrEqual(double)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");  
    else                                  printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchLessOrEqual(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");  
    else                                printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchFirst(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst(100.0)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayDouble::SearchLast(double)  
#include <Arrays\ArrayDouble.mqh>  
///---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLast(100.0)!=-1) printf("Element found");  
    else                           printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    double element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayDouble::SearchLinear(double)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- search element  
    if(array.SearchLinear(100.0) != -1) printf("Element found");  
    else                               printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayDouble::Save(int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(i);  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    ///--- delete array  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayDouble::Load(int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrays elements  
    for(int i=0;i<array.Total();i++)  
    {  
        printf("Element[%d] = %f",i,array.At(i));  
    }  
}
```



```
    }  
    ///--- delete array  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayDouble - 87).

Example:

```
//--- example for CArrayDouble::Type()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayString

CArrayString class is a class of dynamic array of string variables.

Description

The CArrayString class provides the ability to work with a dynamic array of string variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayString : public CArray
```

Title

```
#include <Arrays\ArrayString.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayString

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array opetations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array

Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayString::Reserve(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayString *array=new CArrayString;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size      // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayString::Resize(int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shutdown

Clears the array with a full memory release.

```
bool Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayString::Shutdown()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayString *array=new CArrayString;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    string element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
///--- example for CArrayString::Add(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(IntegerToString(i)))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const string& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayString::AddArray(const string &[])  
#include <Arrays\ArrayString.mqh>  
///---  
string src[];  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements of one array to the end of another.

```
bool AddArray(  
    const CArrayString* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of CArrayString class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
///--- example for CArrayString::AddArray(const CArrayString*)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayString *src=new CArrayString;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- add source arrays elements  
    ///--- . . .  
    ///--- add another array  
    if(!array.AddArray(src))  
    {  
        printf("Array addition error");  
        delete src;  
        delete array;  
        return;  
    }  
    ///--- delete source array  
    delete src;
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    string element,    // element to insert  
    int    pos         // position  
)
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayString::Insert(string,int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(IntegerToString(i),0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    const string& src[],      // source array  
    int          pos         // position  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayString::InsertArray(const string &[],int)  
#include <Arrays\ArrayString.mqh>  
//---  
string src[];  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete array;  
        return;  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    CArrayString* src,      // pointer to the source  
    int          pos       // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayString::InsertArray(const CArrayString*,int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayString *src=new CArrayString;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- insert another array  
    if(!array.InsertArray(src,0))  
    {  
        printf("Array inserting error");  
        delete src;  
        delete array;  
    }  
}
```



```
        return;  
    }  
    //--- delete source array  
    delete src;  
    //--- use array  
    //--- . . .  
    //--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const string& src[]      // source array  
)
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayString::AssignArray(const string &[])  
#include <Arrays\ArrayString.mqh>  
///---  
string src[];  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete array;  
        return;  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AssignArray

Copies the elements of one array to another.

```
bool AssignArray(  
    const CArrayString* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayString::AssignArray(const CArrayString*)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayString *src =new CArrayString;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- add source arrays elements  
    //--- . . .  
    //--- assign another array  
    if(!array.AssignArray(src))  
    {  
        printf("Array assigned error");  
        delete src;  
        delete array;  
        return;  
    }  
    //--- arrays is identical  
    //--- delete source array
```



```
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int    pos,           // position  
    string element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayString::Update(int, string)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,"ABC"))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayString::Shift(int,int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
///--- example for CArrayString::Delete(int)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayString::DeleteRange(int,int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
string At(  
    int pos      // position  
    ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, "" - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, "" may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
///--- example for CArrayString::At(int)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        string result=array.At(i);  
        if(result=="" && GetLastError()==ERR_OUT_OF_RANGE)  
        {  
            ///--- Error reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
}
```



```
//--- delete array  
delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const string& src[]      // source array  
    ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
///--- example for CArrayString::CompareArray(const string &[])  
#include <Arrays\ArrayString.mqh>  
///---  
string src[];  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete array  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArrays (
    const CArrayString* src      // pointer to the source
) const
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
///--- example for CArrayString::CompareArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
///---
void OnStart ()
{
    CArrayString *array=new CArrayString;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- create source array
    CArrayString *src=new CArrayString;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    ///--- add source arrays elements
    ///--- . . .
    ///--- compare with another array
    int result=array.CompareArray(src);
    ///--- delete arrays
    delete src;
    delete array;
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    string element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
///--- example for CArrayString::InsertSort(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort("ABC"))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString::Search(string)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.Search("ABC")!=-1) printf("Element found");  
    else                        printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString::SearchGreat(string)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchGreat("ABC")!= -1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayString:: SearchLess(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLess("ABC")!=-1) printf("Element found");  
    else                          printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    string element    // sample  
) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayString:: SearchGreatOrEqual(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchGreatOrEqual("ABC")!=-1) printf("Element found");  
    else                                     printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayString:: SearchLessOrEqual(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLessOrEqual("ABC")!=-1) printf("Element found");  
    else                                printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchFirst(string)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- search element  
    if(array.SearchFirst("ABC")!= -1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayString:: SearchLast(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- search element  
    if(array.SearchLast("ABC")!= -1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLinear

Searches for the element equal to the sample in the array.

```
int SearchLinear(  
    string element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
///--- example for CArrayString::SearchLinear(string)  
#include <Arrays\ArrayString.mqh>  
///---  
void OnStart()  
{  
    CArrayString *array=new CArrayString;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLinear("ABC")!=-1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayString::Save(int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayString *array=new CArrayString;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        array.Add(IntegerToString(i));  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }
```



```
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(
    int file_handle    // file handle
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayString::Load(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    int file_handle;
    CArrayString *array=new CArrayString;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
    {
        printf("Element[%d] = '%s'",i,array.At(i));
```



```
    }  
    delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayString - 89).

Example:

```
//--- example for CArrayString::Type()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayString *array=new CArrayString;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CArrayObj

CArrayObj class is a class of dynamic array of pointers to instances of CObject and its derived classes.

Description

Class CArrayObj provides the ability to work with a dynamic array of pointers to instances of [CObject](#) and its derived classes. This allows working both with multidimensional dynamic arrays of primitive data types and with data structures that have more complex organization of data.

The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

There are certain [subtleties](#) of the class CArrayObj.

Declaration

```
class CArrayObj : public CArray
```

Title

```
#include <Arrays\ArrayObj.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

CArrayObj

Direct descendants

[CIndicators](#), [CSeries](#)

Class Methods by Groups

Attributes	
FreeMode	Gets the flag of memory management
FreeMode	Sets the flag of memory management
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with full deallocation of its memory (but not its elements).
Creating a new element	
virtual CreateElement	Creates a new array element in the specified position

Add methods	
Add	Adds an element to the end of the array
AddArray	Adds an element to the end of the array
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Detach	Gets the element from the specified position and removes it from the array
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Clear	Removes all elements of the array without the release of the array memory
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the

	sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
Input/output	
Save	Saves data array in the file
Load	Loads data array from the file
Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Arrays of the CObject class have practical application (including all classes of the Standard Library).

For example, consider the options for two-dimensional array:

```
#include <Arrays\ArrayDouble.mqh>
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    int i,j;
    int first_size=10;
    int second_size=100;
    //--- create array
    CArrayObj *array=new CArrayObj;
    CArrayDouble *sub_array;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create subarrays
    for(i=0;i<first_size;i++)
    {
        sub_array=new CArrayDouble;
        if(sub_array==NULL)
```



```

    {
        delete array;
        printf("Object create error");
        return;
    }
    //--- fill array
    for(j=0;j<second_size;j++)
    {
        sub_array.Add(i*j);
    }
    array.Add(sub_array);
}
//--- create array OK
for(i=0;i<first_size;i++)
{
    sub_array=array.At(i);
    for(j=0;j<second_size;j++)
    {
        double element=sub_array.At(j);
        //--- use array element
    }
}
delete array;
}

```

Subtleties

The class has a mechanism to control dynamic memory, so be careful when working with elements of the array.

Mechanism of memory management can be switched on/off using the method FreeMode (bool). By default, the mechanism is enabled.

Accordingly, there are two options for dealing with the CArrayObj class:

1. Mechanism of memory management is enabled. (default)

In this case, CArrayObj takes responsibility for releasing the memory used for the elements after their removal from the array. A custom program should not release the array elements.

Example:

```

int i;
//--- create an array
CArrayObj *array=new CArrayObj;
//--- fill array elements
for(i=0;i<10;i++) array.Add(new CObject);
//--- do something
for(i=0;i<array.Total();i++)
{

```



```
    CObject *object=array.At(i);  
    //--- actions performed with the element  
    . . .  
}  
//--- remove the array with the elements  
delete array;
```

2. Mechanism of memory management is disabled.

In this case, CArrayObj is not responsible for deallocating of the elements' memory after their removal from the array. Besides, the user program must deallocate the array elements.

Example:

```
int i;  
//--- create an array  
CArrayObj *array=new CArrayObj;  
//--- disable the mechanism of memory management  
array.FreeMode(false);  
//--- fill array with elements  
for(i=0;i<10;i++) array.Add(new CObject);  
//--- do something  
for(i=0;i<array.Total();i++)  
{  
    CObject *object=array.At(i);  
    //--- actions performed with the element  
    . . .  
}  
//--- remove array elements  
while(array.Total()) delete array.Detach();  
//--- remove empty array  
delete array;
```


FreeMode

Gets the flag of memory management.

```
bool FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CArrayObj::FreeMode()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get free mode flag
    bool array_free_mode=array.FreeMode();
    //--- delete array
    delete array;
}
```


FreeMode

Sets the flag of memory management.

```
void FreeMode(  
    bool mode    // new flag  
)
```

Parameters

mode

[in] New value of the memory management flag.

Return Value

None.

Note

Setting the memory management flag is an important part in the CArrayObj class use. Since the array elements are pointers to dynamic objects, it is important to determine what to do with them when removing from the array.

If the flag is set, removing an element from the array, the element is automatically deleted by the delete operator. If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be deallocated by the program afterwards.

If the user program resets the flag of memory management, the user must understand his responsibility for the removal of the array before the termination of the program. Otherwise the memory allocated for elements by the new operator is not released.

For large amounts of data this could lead even to crash of your terminal.

If the user does not reset the memory management flag, there is another pitfall. When pointer elements in array are stored somewhere in the local variables, then removing the array will lead to a critical error and crash of the user program. By default, the memory management flag is set, i.e. the class of the array is responsible for freeing the memory elements.

Example:

```
//--- example for CArrayObj::FreeMode(bool)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- reset free mode flag  
    array.FreeMode(false);
```



```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Reserve

Allocates memory to increase the size of the array.

```
bool Reserve (
    int size      // number
)
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
///--- example for CArrayObj::Reserve(int)
#include <Arrays\ArrayObj.mqh>
///---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    ///---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    ///--- use array
    ///--- . . .
    ///--- delete array
    delete array;
}
```


Resize

Sets a new (smaller) size of the array.

```
bool Resize(  
    int size    // size  
)
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than or equal to zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. The memory of the lost elements is released or not depending on the memory management mode.

To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
///--- example for CArrayObj::Resize(int)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- resize array  
    if(!array.Resize(10))  
    {  
        printf("Resize error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Clear

Removes all elements of the array without the release of the memory array.

```
void Clear()
```

Return Value

No.

Note

If the memory management flag is enabled, the memory used for the deleted objects is released.

Example:

```
//--- example for CArrayObj::Clear()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- clear array
    array.Clear();
    //--- delete array
    delete array;
}
```


Shutdown

Clears the array with full deallocation of memory for it (but not for its elements).

```
bool Shutdown()
```

Return Value

true - successful, false - an error occurred.

Note

If memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CArrayObj::Shutdown()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```


CreateElement

Creates a new array element at the specified position.

```
bool CreateElement(  
    int index    // position  
)
```

Parameters

index

[in] Position in which you want to create a new element.

Return Value

true - successful, false - cannot create an element.

Note

Method CreateElement (int) in class CArrayObj always returns false and does not perform any action. If necessary, the CreateElement(int) method should be implemented in a derived class.

Example:

```
///--- example for CArrayObj::CreateElement(int)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    int size=100;  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- fill array  
    array.Reserve(size);  
    for(int i=0;i<size;i++)  
    {  
        if(!array.CreateElement(i))  
        {  
            printf("Element create error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;
```



```
}
```


Add

Adds an element to the end of the array.

```
bool Add(  
    CObject* element    // element to add  
)
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
///--- example for CArrayObj::Add(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add 100 arrays elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Add(new CObject))  
        {  
            printf("Element addition error");  
            delete array;  
            return;  
        }  
    }  
    ///--- use array  
    ///--- . . .  
    ///--- delete array  
    delete array;  
}
```


AddArray

Adds elements from one array to the end of another.

```
bool AddArray(  
    const CArrayObj * src      // pointer to the source array  
)
```

Parameters

src

[in] Pointer to an instance of the [CArrayDouble](#) class - source of elements to add.

Return Value

true - successful, false - cannot add items.

Note

Adding elements from array to array is actually copying the pointers. Therefore, when calling the method, there is a pitfall - there may be a pointer to a dynamic object in more than one variable.

```
//--- example  
extern bool      make_error;  
extern int       error;  
extern CArrayObj *src;  
//--- create a new instance CArrayObj  
//--- default memory management is turned on  
CArrayObj *array=new CArrayObj;  
//--- add (copy) the elements from the source array  
if(array!=NULL)  
    bool result=array.AddArray(src);  
if(make_error)  
{  
    //--- perform erroneous actions  
    switch(error)  
    {  
        case 0:  
            //--- remove the source array without checking its memory management flag  
            delete src;  
            //--- result:  
            //--- it is possible to address an element by invalid pointer in the receiver  
            break;  
        case 1:  
            //--- disable the mechanism of memory management in the source array  
            if(src.FreeMode()) src.FreeMode(false);  
            //--- but do not remove the source array  
            //--- result:  
            //--- after removing the receiver array, it is possible to address an element  
            break;  
        case 2:
```



```

        //--- disable the mechanism of memory management in the source array
        src.FreeMode(false);
        //--- disable the mechanism of memory management in the receiver array
        array.FreeMode(false);
        //--- result:
        //--- after the program termination, get a "memory leak"
        break;
    }
}
else
{
    //--- disable the mechanism of memory management in the source array
    if(src.FreeMode()) src.FreeMode(false);
    //--- delete the source array
    delete src;
    //--- result:
    //--- addressing the receiver array element will be correct
    //--- deleting the receiver array will lead to deleting its elements
}

```

Example:

```

//--- example for CArrayObj::AddArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayObj *src=new CArrayObj;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- reset free mode flag
    src.FreeMode(false);
    //--- fill source array
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))

```



```
{  
    printf("Array addition error");  
    delete src;  
    delete array;  
    return;  
}  
//--- delete source array without elements  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```


Insert

Inserts an element to the specified position in the array.

```
bool Insert(  
    CObject* element,    // element to insert  
    int      pos         // position  
)
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
//--- example for CArrayObj::Insert(CObject*,int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert elements  
    for(int i=0;i<100;i++)  
    {  
        if(!array.Insert(new CObject,0))  
        {  
            printf("Insert error");  
            delete array;  
            return;  
        }  
    }  
    //--- use array  
    //--- . . .  
    //--- delete array
```



```
delete array;  
}
```


InsertArray

Inserts elements of one array from the specified position of another array.

```
bool InsertArray(  
    const CArrayObj* src,      // pointer to the source  
    int pos                   // position  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Note

See: [CArrayObj::AddArray\(const CArrayObj*\)](#).

Example:

```
//--- example for CArrayObj::InsertArray(const CArrayObj*,int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayObj *src=new CArrayObj;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- reset free mode flag  
    src.FreeMode(false);  
    //--- fill source array  
    //--- . . .  
    //--- insert another array
```



```
if(!array.InsertArray(src,0))
{
    printf("Array inserting error");
    delete src;
    delete array;
    return;
}
//--- delete source array without elements
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```


AssignArray

Compares the array with another one.

```
bool AssignArray(  
    const CArrayObj* src      // pointer to the source  
)
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Note

If the receiver array is not empty when calling AssignArray, then all its elements will be removed; and if the memory management flag is set, the memory used for the deleted elements will be released. The receiver array becomes an exact copy of the source one. Additionally, see [CArrayObj::AddArray\(const CArrayObj*\)](#).

Example:

```
//--- example for CArrayObj::AssignArray(const CArrayObj*)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source array  
    CArrayObj *src=new CArrayObj;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    //--- reset free mode flag  
    src.FreeMode(false);  
    //--- fill source array  
    //--- . . .  
    //--- assign another array  
    if(!array.AssignArray(src))
```



```
{
    printf("Array assigned error");
    delete src;
    delete array;
    return;
}

//--- arrays is identical
//--- delete source array without elements
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```


Update

Changes the element at the specified array position.

```
bool Update(  
    int      pos,          // position  
    CObject* element       // value  
)
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Note

The element will not change if an invalid pointer (for example, NULL) is passed as a parameter. If the memory management is enabled, the memory of a replaced element is deallocated.

Example:

```
//--- example for CArrayObj::Update(int,CObject*)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- update element  
    if(!array.Update(0,new CObject))  
    {  
        printf("Update error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Shift

Moves an item from a given position in the array to the specified offset.

```
bool Shift(  
    int pos,          // position  
    int shift         // shift  
)
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayObj::Shift(int,int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- shift element  
    if(!array.Shift(10,-5))  
    {  
        printf("Shift error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```


Detach

Removes an element from a given position in the array.

```
CObject* Detach(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of a removed item in the array.

Return Value

Pointer to the removed element - success, NULL - cannot remove the element.

Note

When an element is removed from the array, it will not be deleted regardless of the memory management flag. Once the array element pointer is used, it has to be deallocated.

Example:

```
//--- example for CArrayObj::Detach(int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    CObject *object=array.Detach(0);  
    if(object==NULL)  
    {  
        printf("Detach error");  
        delete array;  
        return;  
    }  
    //--- use element  
    //--- . . .  
    //--- delete element  
    delete object;  
    //--- delete array  
    delete array;  
}
```


Delete

Removes the element from the specified array position.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
///--- example for CArrayObj::Delete(int)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


DeleteRange

Deletes a group of elements from the specified array position.

```
bool DeleteRange(  
    int from,      // position of the first element  
    int to         // position of last element  
)
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove elements.

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
///--- example for CArrayObj::DeleteRange(int,int)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- delete elements  
    if(!array.DeleteRange(0,10))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


At

Gets the element from the specified array position.

```
CObject* At(  
    int pos    // position  
)
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - successful, NULL- there was an attempt to get an element of a non-existent position.

Example:

```
///--- example for CArrayObj::At(int)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add elements  
    ///--- . . .  
    for(int i=0;i<array.Total();i++)  
    {  
        CObject *result=array.At(i);  
        if(result==NULL)  
        {  
            ///--- Error reading from array  
            printf("Get element error");  
            delete array;  
            return;  
        }  
        ///--- use element  
        ///--- . . .  
    }  
    delete array;  
}
```


CompareArray

Compares the array with another one.

```
bool CompareArray(  
    const CArrayObj* src      // pointer to the source  
    ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements for comparison.

Return Value

true - the arrays are equal - the arrays are not equal.

Example:

```
///--- example for CArrayObj::CompareArray(const CArrayObj*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- create source array  
    CArrayObj *src=new CArrayObj;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete array;  
        return;  
    }  
    ///--- fill source array  
    ///--- . . .  
    ///--- compare with another array  
    int result=array.CompareArray(src);  
    ///--- delete arrays  
    delete src;  
    delete array;  
}
```


InsertSort

Inserts an element in a sorted array.

```
bool InsertSort(  
    CObject* element    // element to insert  
)
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
///--- example for CArrayObj::InsertSort(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- insert element  
    if(!array.InsertSort(new CObject))  
    {  
        printf("Insert error");  
        delete array;  
        return;  
    }  
    ///--- delete array  
    delete array;  
}
```


Search

Searches for an element equal to the sample in the sorted array.

```
int Search(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::Search(CObject*)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    //--- set sample attributes  
    //--- . . .  
    //--- search element  
    if(array.Search(sample)!=-1) printf("Element found");  
    else                        printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int SearchGreat(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayObj::SearchGreat(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    ///--- set sample attributes  
    ///--- . . .  
    ///--- search element  
    if(array.SearchGreat(sample)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int SearchLess(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayObj:: SearchLess(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    ///--- set sample attributes  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLess(sample)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int SearchGreatOrEqual(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::SearchGreatOrEqual(CObject*)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    //--- set sample attributes  
    //--- . . .  
    //--- search element  
    if(array.SearchGreatOrEqual(sample)!=-1) printf("Element found");  
    else                                     printf("Element not found");  
    //--- delete array  
    delete array;
```



```
}
```


SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int SearchLessOrEqual(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayObj:: SearchLessOrEqual(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    ///--- set sample attributes  
    ///--- . . .  
    ///--- search element  
    if(array.SearchLessOrEqual(sample)!=-1) printf("Element found");  
    else                                  printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int SearchFirst(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
///--- example for CArrayObj::SearchFirst(CObject*)  
#include <Arrays\ArrayObj.mqh>  
///---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    ///---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add arrays elements  
    ///--- . . .  
    ///--- sort array  
    array.Sort();  
    ///--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    ///--- set sample attributes  
    ///--- . . .  
    ///--- search element  
    if(array.SearchFirst(sample)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    ///--- delete array  
    delete array;  
}
```


SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int SearchLast(  
    CObject* element    // sample  
    ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj:: SearchLast(CObject*)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- sort array  
    array.Sort();  
    //--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete array;  
        return;  
    }  
    //--- set sample attributes  
    //--- . . .  
    //--- search element  
    if(array.SearchLast(sample)!=-1) printf("Element found");  
    else                             printf("Element not found");  
    //--- delete array  
    delete array;  
}
```


Save

Saves data array in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayObj::Save(int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    delete array;  
}
```


Load

Loads data array from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Note

When reading array elements from the file, the [CArrayObj::CreateElement\(int\)](#) method is called to create each element.

Example:

```
//--- example for CArrayObj::Load(int)  
#include <Arrays\ArrayObj.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CArrayObj *array=new CArrayObj;  
    //---  
    if(array!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!array.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete array;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
}
```



```
//--- use arrays elements  
//--- . . .  
//--- delete array  
delete array;  
}
```


Type

Gets the array type identifier.

```
virtual int Type() const
```

Return Value

Array type identifier (for CArrayObj - 7778).

Example:

```
//--- example for CArrayObj::Type()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```


CList

CList Class is a class of dynamic list of instances of the CObject class and its derived classes.

Description

Class CList provides the ability to work with a list of instances of [CObject](#) and its derived classes. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

There are some subtleties of working with the CList class. The class has a mechanism to control dynamic memory, so be careful when working with elements of the list.

[Subtleties](#) of the mechanism of memory management similar to those described in CArrayObj.

Declaration

```
class CList : public CObject
```

Title

```
#include <Arrays\List.mqh>
```

Inheritance hierarchy

[CObject](#)

CList

Class Methods by Groups

Attributes	
FreeMode	Gets the flag of memory management when deleting list elements
FreeMode	Sets the flag of memory management when deleting list elements
Total	Gets the number of elements in the list
IsSorted	Gets sorted list flag
SortMode	Gets the sorting mode
Create methods	
CreateElement	Creates a new list element
Add methods	
Add	Adds an element to the end of the list
Insert	Inserts an element to the specified position of the list

Delete methods	
DetachCurrent	Removes an element from the current position in the list without deleting it "physically"
DeleteCurrent	Removes the element from the current position in the list
Delete	Removes the element from the specified position in the list
Clear	Removes all list elements
Navigation	
IndexOf	Gets the index of the specified list element
GetNodeAtIndex	Gets an item with the specified index of the list
GetFirstNode	Gets the first element of the list
GetPrevNode	Gets the previous element of the list
GetCurrentNode	Gets the current list element
GetNextNode	Gets the next element in the list
GetLastNode	Gets the last element in the list
Ordering methods	
Sort	Sorts the list
MoveToIndex	Moves the current element in the list to the specified position
Exchange	Swaps two elements in the list
Compare methods	
CompareList	Compares the list with another one
Search methods	
Search	Searches for an element equal to the sample in sorted list
Input/output	
virtual Save	Saves list data in the file
virtual Load	Loads list data from the file
virtual Type	Gets the list type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

FreeMode

Gets the flag of memory management when deleting list elements.

```
bool FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CList::FreeMode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get free mode flag
    bool list_free_mode=list.FreeMode();
    //--- delete list
    delete list;
}
```


FreeMode

Sets the flag of memory management when deleting list elements.

```
void FreeMode(  
    bool mode    // new value  
)
```

Parameters

mode

[in] New value of the memory management flag.

Note

Setting the memory management flag is an important part in the CList class use. Since the list elements are pointers to dynamic objects, it is important to determine what to do with them when removing from the list.

If the flag is set, when removing an element from the list, the element is automatically deleted by the delete operator. If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be deallocated by the program afterwards.

If the user program resets the flag of memory management, users should understand their responsibility for the removal of the list elements before the termination of the program. Otherwise, the memory allocated for elements by the new operator is not released.

For large amounts of data this could lead even to a terminal crash.

If the user program does not reset the memory management flag, there is another "pitfall". When pointer elements in list are stored somewhere in the local variables, then removing the list will lead to a critical error and crash of the user program. By default, the memory management flag is set, i.e. the class of the list is responsible for releasing the memory elements.

Example:

```
//--- example for CList::FreeMode(bool)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- reset free mode flag  
    list.FreeMode(false);  
    //--- use list  
    //--- . . .  
    //--- delete list
```



```
delete list;  
}
```


Total

Gets the number of elements in the list.

```
int Total() const
```

Return Value

Number of elements in the list.

Example:

```
//--- example for CList::Total()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check total
    int total=list.Total();
    //--- use list
    //--- ...
    //--- delete list
    delete list;
}
```


IsSorted

Gets the sorted list flag.

```
bool IsSorted(  
    int mode=0      // sorting mode  
) const
```

Parameters

mode=0

[in] Checked sort mode.

Return Value

Flag of the sorted list. Returns true if the list is sorted using the specified mode, otherwise returns false.

Note

Flag of the sorted list cannot be changed directly. The flag is set by Sort(int) and resets by any add/insert methods.

Example:

```
///--- example for CList::IsSorted()  
#include <Arrays\List.mqh>  
///---  
void OnStart()  
{  
    CList *list=new CList;  
    ///---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- check sorted  
    if(list.IsSorted(0))  
    {  
        ///--- use methods for sorted list  
        ///--- ...  
    }  
    ///--- delete list  
    delete list;  
}
```


SortMode

Gets the version of the sorting.

```
int SortMode() const
```

Return Value

Sorting mode, or -1 if the list is not sorted.

Example:

```
//--- example for CList::SortMode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sort mode
    int sort_mode=list.SortMode();
    //--- use list
    //--- ...
    //--- delete list
    delete list;
}
```


CreateElement

Creates a new element of the list.

```
CObject* CreateElement()
```

Return Value

Pointer to the newly created element - successful, NULL - cannot create an element.

Note

Method `CreateElement()` in the `CList` class always returns `NULL` and does not perform any actions. If necessary, method `CreateElement()` should be implemented in a derived class.

Example:

```
//--- example for CList::CreateElement(int)
#include <Arrays\List.mqh>
//---
void OnStart()
{
    int    size=100;
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- fill list
    for(int i=0;i<size;i++)
    {
        CObject *object=list.CreateElement();
        if(object==NULL)
        {
            printf("Element create error");
            delete list;
            return;
        }
        list.Add(object);
    }
    //--- use list
    //--- . . .
    //--- delete list
    delete list;
}
```


Add

Adds an element to the end of the list.

```
int Add(  
    CObject* element    // element to add  
)
```

Parameters

element

[in] Value of the element to add to the list.

Return Value

Index of the added element - success, -1 - error.

Note

The element is not added to the list, if an invalid pointer (for example, NULL) is passed as a parameter.

Example:

```
///--- example for CList::Add(CObject*)  
#include <Arrays\List.mqh>  
///---  
void OnStart()  
{  
    CList *list=new CList;  
    ///---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add 100 elements  
    for(int i=0;i<100;i++)  
    {  
        if(list.Add(new CObject)==-1)  
        {  
            printf("Element addition error");  
            delete list;  
            return;  
        }  
    }  
    ///--- use list  
    ///--- . . .  
    ///--- delete list  
    delete list;  
}
```


Insert

Inserts an element to the specified position in the list.

```
int Insert(  
    CObject* element,    // element to insert  
    int      pos         // position  
)
```

Parameters

element

[in] value of the element to insert in the list

pos

[in] position in the list to insert

Return Value

index of inserted element - success, -1 - error.

Note

The element is not added to the list, if an invalid pointer (for example, NULL) is passed as a parameter.

Example:

```
//--- example for CList::Insert(CObject*,int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- insert 100 elements  
    for(int i=0;i<100;i++)  
    {  
        if(list.Insert(new CObject,0)==-1)  
        {  
            printf("Element insert error");  
            delete list;  
            return;  
        }  
    }  
    //--- use list  
    //--- . . .
```



```
//--- delete list  
delete list;  
}
```


DetachCurrent

Extracts an element from the current position in the list without its "physical" deletion.

```
CObject* DetachCurrent()
```

Return Value

Pointer to the removed element in case of success, NULL - cannot remove the element.

Note

When removed from the list, the element is not removed in any state of the memory management flag. The pointer to the extracted element should be released after it has been used.

Example:

```
//--- example for CList::DetachCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.DetachCurrent();
    if(object==NULL)
    {
        printf("Detach error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- delete element
    delete object;
    //--- delete list
    delete list;
}
```


DeleteCurrent

Removes the element from the current position in the list.

```
bool DeleteCurrent()
```

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management is enabled, memory for the removed element is deallocated.

Example:

```
///--- example for CList::DeleteCurrent()
#include <Arrays\List.mqh>
///---
void OnStart()
{
    CList *list=new CList;
    ///---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    ///--- add list elements
    ///--- . . .
    if(!list.DeleteCurrent())
    {
        printf("Delete error");
        delete list;
        return;
    }
    ///--- delete list
    delete list;
}
```


Delete

Removes the element from the given position in the list.

```
bool Delete(  
    int pos    // position  
)
```

Parameters

pos

[in] position of element to be removed from the list.

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management flag is enabled, the memory used for the deleted element is released.

Example:

```
///--- example for CList::Delete(int)  
#include <Arrays\List.mqh>  
///---  
void OnStart()  
{  
    CList *list=new CList;  
    ///---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add list elements  
    ///--- . . .  
    if(!list.Delete(0))  
    {  
        printf("Delete error");  
        delete list;  
        return;  
    }  
    ///--- delete list  
    delete list;  
}
```


Clear

Removes all elements of the list.

```
void Clear()
```

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CList::Clear()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    //--- clear list
    list.Clear();
    //--- delete list
    delete list;
}
```


IndexOf

Gets the index of the specified list element.

```
int IndexOf(  
    CObject* element    // pointer to the element  
)
```

Parameters

element

[in] pointer to the list element.

Return Value

List element index, or -1.

Example:

```
///--- example for CList::IndexOf(CObject*)  
#include <Arrays\List.mqh>  
///---  
void OnStart()  
{  
    CList *list=new CList;  
    ///---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    CObject *object=new CObject;  
    if(object==NULL)  
    {  
        printf("Element create error");  
        delete list;  
        return;  
    }  
    if(list.Add(object))  
    {  
        int pos=list.IndexOf(object);  
    }  
    ///--- delete list  
    delete list;  
}
```


GetNodeAtIndex

Gets an element from the specified position in the list.

```
CObject* GetNodeAtIndex(  
    int pos      // position  
)
```

Parameters

pos

[in] element position in the list.

Return Value

pointer to the element - success, NULL - cannot receive a pointer.

Example:

```
//--- example for CList::GetNodeAtIndex(int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add list elements  
    //--- . . .  
    CObject *object=list.GetNodeAtIndex(10);  
    if(object==NULL)  
    {  
        printf("Get node error");  
        delete list;  
        return;  
    }  
    //--- use element  
    //--- . . .  
    //--- do not delete element  
    //--- delete list  
    delete list;  
}
```


GetFirstNode

Gets the first element of the list.

```
CObject* GetFirstNode()
```

Return Value

Pointer to the first element - success, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetFirstNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetFirstNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```


GetPrevNode

Gets the previous element of the list.

```
CObject* GetPrevNode()
```

Return Value

Pointer to the previous element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetPrevNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetPrevNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```


GetCurrentNode

Gets the current list element.

```
CObject* GetCurrentNode()
```

Return Value

Pointer to the current element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetCurrentNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetCurrentNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```


GetNextNode

Gets the next element in the list.

```
CObject* GetNextNode()
```

Return Value

Pointer to the next element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetNextNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetNextNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```


GetLastNode

Gets the last element of the list.

```
CObject* GetLastNode()
```

Return Value

Pointer to the last element - success, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetLastNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetLastNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```


Sort

Sorts a list.

```
void Sort(  
    int mode    // sorting mode  
)
```

Parameters

mode

[in] Sorting mode.

Return Value

No.

Note

The list is always sorted in ascending order.

Example:

```
//--- example for CList::Sort(int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- sorting by mode 0  
    list.Sort(0);  
    //--- use list  
    //--- ...  
    //--- delete list  
    delete list;  
}
```


MoveToIndex

Moves the current element in the list to the specified position.

```
bool MoveToIndex(  
    int pos    // position  
)
```

Parameters

pos

[in] position in the list to move.

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CList::MoveToIndex(int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- move current node to begin  
    list.MoveToIndex(0);  
    //--- use list  
    //--- . . .  
    //--- delete list  
    delete list;  
}
```


Exchange

Swaps two elements in the list.

```
bool Exchange(  
    CObject* node1,    // list element  
    CObject* node2     // list element  
)
```

Parameters

node1

[in] list element

node2

[in] list element

Return Value

true - successful, false - cannot swap the elements.

Example:

```
//--- example for CList::Exchange(CObject*,CObject*)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- exchange  
    list.Exchange(list.GetFirstNode(),list.GetLastNode());  
    //--- use list  
    //--- . . .  
    //--- delete list  
    delete list;  
}
```


CompareList

Compares the list with another one.

```
bool CompareList(  
    CList* list      // pointer to the source  
)
```

Parameters

list

[in] a pointer to an instance of the CList class used as a source of elements for comparison.

Return Value

true - the lists are equal, false - the lists are not equal.

Example:

```
//--- example for CList::CompareList(const CList*)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- create source list  
    CList *src=new CList;  
    if(src==NULL)  
    {  
        printf("Object create error");  
        delete list;  
        return;  
    }  
    //--- fill source list  
    //--- . . .  
    //--- compare with another list  
    bool result=list.CompareList(src);  
    //--- delete lists  
    delete src;  
    delete list;  
}
```


Search

Searches for an element equal to the sample in the sorted list.

```
CObject* Search(  
    CObject* element    // sample  
)
```

Parameters

element

[in] element sample to search for in the list.

Return Value

Pointer to the found element - successful, NULL - the element is not found.

Example:

```
//--- example for CList::Search(CObject*)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add lists elements  
    //--- . . .  
    //--- sort list  
    list.Sort(0);  
    //--- create sample  
    CObject *sample=new CObject;  
    if(sample==NULL)  
    {  
        printf("Sample create error");  
        delete list;  
        return;  
    }  
    //--- set sample attributes  
    //--- . . .  
    //--- search element  
    if(list.Search(sample)!=NULL) printf("Element found");  
    else                          printf("Element not found");  
    //--- delete list  
    delete list;  
}
```


Save

Saves list data in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Example:

```
///--- example for CList::Save(int)  
#include <Arrays\List.mqh>  
///---  
void OnStart()  
{  
    int file_handle;  
    CList *list=new CList;  
    ///---  
    if(list!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    ///--- add lists elements  
    ///--- . . .  
    ///--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!list.Save(file_handle))  
        {  
            ///--- file save error  
            printf("File save: Error %d!",GetLastError());  
            delete list;  
            FileClose(file_handle);  
            ///---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    ///--- delete list  
    delete list;
```



```
}
```


Load

Loads list data from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Note

When reading list elements from the file, the [CList::CreateElement\(int\)](#) method is called to create each element.

Example:

```
//--- example for CLoad::Load(int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CList *list=new CList;  
    //---  
    if(list!=NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!list.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            delete list;  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
}
```



```
//--- use list elements  
//--- . . .  
//--- delete list  
delete list;  
}
```


Type

Gets the list type identifier.

```
virtual int Type()
```

Return Value

List type identifier (for CList - 7779).

Example:

```
//--- example for CList::Type()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get list type
    int type=list.Type();
    //--- delete list
    delete list;
}
```


CTreeNode

CTreeNode is a class of the CTree binary tree node.

Description

CTreeNode provides the ability to work with nodes of the [CTree](#) binary tree. Options of navigation through the tree are implemented in the class. Besides, methods of working with the file are implemented.

Declaration

```
class CTreeNode : public CObject
```

Title

```
#include <Arrays\TreeNode.mqh>
```

Inheritance hierarchy

[CObject](#)

CTreeNode

Direct descendants

[CTree](#)

Class Methods by Groups

Attributes	
Owner	Gets/sets the pointer of the owner node
Left	Gets/sets the pointer of the left node
Right	Gets/sets the pointer of the right node
Balance	Gets the node balance
BalanceL	Gets the balance of the left sub-branch of the node
BalanceR	Gets the balance of the right sub-branch of the node
Creation of a new element	
CreateSample	Creates a new node instance
Comparison	
RefreshBalance	Recalculates the node balance
Search	
GetNext	Gets the pointer of the next node

Input/Output	
SaveNode	Saves the node data to a file
LoadNode	Downloads the node data from a file
virtual Type	Gets the identifier of the node type

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Compare](#)

Trees of CTreeNode class descendants have practical application.

A descendant of CTreeNode class should have predefined methods: [CreateSample](#) creates a new instance of the descendant class of CTreeNode, [Compare](#) compares values of key fields of the descendant class of CTreeNode, [Type](#) (if it is necessary to identify a node), [SaveNode](#) and [LoadNode](#) (if it is necessary to work with a file).

Let's consider an example of a CTree descendant class.

```
//+-----+
//|                                     MyTreeNode.mq5 |
//|                                     Copyright 2010, MetaQuotes Software Corp. |
//|                                     https://www.metaquotes.net/ |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\TreeNode.mqh>
//+-----+
//| Describe CMyTreeNode class derived from CTreeNode. |
//+-----+
//| Class CMyTreeNode. |
//| Purpose: Class of element of a binary tree. |
//|           Descendant of class CTreeNode. |
//+-----+
class CMyTreeNode : public CTreeNode
{
protected:
    //--- user's data
    long      m_long;           // key field of long type
    double    m_double;        // custom variable of double type
    string    m_string;        // custom variable of string type
    datetime  m_datetime;      // custom variable of datetime type

public:
    CMyTreeNode();

    //--- methods of accessing user's data
    long      GetLong(void)      { return(m_long); }
    void      SetLong(long value) { m_long=value; }
```



```

double      GetDouble(void)          { return(m_double); }
void        SetDouble(double value)  { m_double=value; }
string      GetString(void)          { return(m_string); }
void        SetString(string value)  { m_string=value; }
datetime    GetDateTime(void)        { return(m_datetime); }
void        SetDateTime(datetime value) { m_datetime=value; }

//--- methods for working with files
virtual bool Save(int file_handle);
virtual bool Load(int file_handle);
protected:
virtual int  Compare(const CObject *node,int mode);
//--- method of creating class instances
virtual CTreeNode* CreateSample();
};

//+-----+
//| CMyTreeNode class constructor. |
//| INPUT:  none.                  |
//| OUTPUT: none.                  |
//| REMARK: none.                  |
//+-----+
void CMyTreeNode::CMyTreeNode()
{
//--- initialization of user's data
    m_long      =0;
    m_double     =0.0;
    m_string     ="";
    m_datetime   =0;
}

//+-----+
//| Comparison with another tree node by the specified algorithm. |
//| INPUT:  node - tree element to compare,                        |
//|          mode - identifier of comparison algorithm.             |
//| OUTPUT: result of comparison (>0,0,<0).                         |
//| REMARK: none.                                                  |
//+-----+
int CMyTreeNode::Compare(const CObject *node,int mode)
{
//--- mode parameter is ignored, because tree construction algorithm is the only one
    int res=0;
//--- explicit type casting
    CMyTreeNode *n=node;
    res=(int) (m_long-n.m_long);
//---
    return(res);
}

//+-----+
//| Creation of a new class instance. |
//| INPUT:  none.                    |
//| OUTPUT: pointer to a new instance of CMyTreeNode class. |

```



```

//| REMARK: none. |
//+-----+
CTreeNode* CMyTreeNode::CreateSample()
{
    CMyTreeNode *result=new CMyTreeNode;
//---
    return(result);
}
//+-----+
//| Write tree node data to a file. |
//| INPUT:  file_handle -handle of a file pre-opened for writing. |
//| OUTPUT: true if OK, otherwise false. |
//| REMARK: none. |
//+-----+
bool CMyTreeNode::Save(int file_handle)
{
    uint i=0,len;
//--- checks
    if(file_handle<0) return(false);
//--- writing user data
//--- writing custom variable of long type
    if(FileWriteLong(file_handle,m_long)!=sizeof(long)) return(false);
//--- writing custom variable of double type
    if(FileWriteDouble(file_handle,m_double)!=sizeof(double)) return(false);
//--- writing custom variable of string type
    len=StringLen(m_string);
//--- write string length
    if(FileWriteInteger(file_handle,len,INT_VALUE)!=INT_VALUE) return(false);
//--- write the string
    if(len!=0 && FileWriteString(file_handle,m_string,len)!=len) return(false);
//--- writing custom variable of datetime type
    if(FileWriteLong(file_handle,m_datetime)!=sizeof(long)) return(false);
//---
    return(true);
}
//+-----+
//| Read tree node data from a file. |
//| INPUT:  file_handle -handle of a file pre-opened for reading. |
//| OUTPUT: true if OK, otherwise false. |
//| REMARK: none. |
//+-----+
bool CMyTreeNode::Load(int file_handle)
{
    uint i=0,len;
//--- checks
    if(file_handle<0) return(false);
//--- reading
    if(FileIsEnding(file_handle)) return(false);
//--- reading custom variable of char type

```



```
//--- reading custom variable of long type
    m_long=FileReadLong(file_handle);
//--- reading custom variable of double type
    m_double=FileReadDouble(file_handle);
//--- reading custom variable of string type
//--- read the string length
    len=FileReadInteger(file_handle,INT_VALUE);
//--- read the string
    if(len!=0) m_string=FileReadString(file_handle,len);
    else      m_string="";
//--- reading custom variable of datetime type
    m_datetime=FileReadLong(file_handle);
//---
    return(true);
}
```


Owner

Gets the pointer of the owner node.

```
CTreeNode* Owner()
```

Return Value

Pointer of the owner node.

Owner

Sets the pointer of the owner node.

```
void Owner(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] New value of the pointer of the owner node.

Return Value

None.

Left

Gets the pointer of the left node.

```
CTreeNode* Left()
```

Return Value

Pointer of the left node.

Left

Sets the pointer of the left node.

```
void Left(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] New value of the pointer of the left node.

Return Value

None.

Right

Gets the pointer of the right node.

```
CTreeNode* Right()
```

Return Value

The pointer of the right node.

Right

Sets the pointer of the right node.

```
void Right(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] New value of the pointer of the right node.

Return Value

None.

Balance

Gets the node balance.

```
int Balance() const
```

Return Value

Node balance.

BalanceL

Gets the balance of the left sub-branch of the node.

```
int BalanceL() const
```

Return Value

Balance of the left sub-branch of the node.

BalanceR

Gets the balance of the right sub-branch of the node.

```
int BalanceR() const
```

Return Value

Balance of the right sub-branch of the node.

CreateSample

Creates a new node sample.

```
virtual CTreeNode* CreateSample()
```

Return Value

Pointer to the new node sample or NULL.

RefreshBalance

Recalculates the node balance.

```
int RefreshBalance()
```

Return Value

Node balance.

GetNext

Gets the pointer of the next node.

```
CTreeNode* GetNext (  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] Node of the search start.

Return Value

Pointer of the next node.

SaveNode

Writes node data to a file.

```
bool SaveNode(  
    int file_handle    // handle  
)
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true - success, otherwise false.

LoadNode

Reads node data from a file.

```
bool LoadNode(  
    int      file_handle,    // handle  
    CTreeNode* main          // node  
)
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

main

[in] Node for data.

Return Value

true - success, otherwise false.

Type

Gets the identifier of the node type.

```
virtual int Type() const
```

Return Value

Identifier of the node type.

CTree

CTree is a class of the binary tree of the instances of CTreeNode class and its descendants.

Description

CTree class provides the possibility to work with the binary tree of [CTreeNode](#) class instances and its descendants. Options of adding/inserting/deleting of tree elements and search in the tree are implemented in the class. Besides that, methods of working with a file are implemented.

Note that mechanism of dynamic memory management is not implemented in class CTree (unlike classes [CList](#) and [CArrayObj](#)). All tree nodes are deleted with memory deallocation.

Declaration

```
class CTree : public CTreeNode
```

Title

```
#include <Arrays\Tree.mqh>
```

Inheritance hierarchy

[CObject](#)

[CTreeNode](#)

CTree

Class Methods by Groups

Attributes	
Root	Gets the root node of the tree
Creation of a new element	
CreateElement	Creates a new instance of the node
Filling	
Insert	Adds a node to a tree
Deletion	
Detach	Detaches a specified node from a tree
Delete	Deletes a specified node from a tree
Clear	Deletes all nodes of a tree
Search	
Find	Searches for a node in a tree by sample
Input/output	
virtual Save	Saves all the tree data to a file

virtual Load	Downloads tree data from a file
virtual Type	Gets identifier of the tree type

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CTreeNode

Parent, Parent, [Left](#), [Left](#), [Right](#), [Right](#), [Balance](#), [BalanceL](#), [BalanceR](#), [RefreshBalance](#), [GetNext](#), [SaveNode](#), [LoadNode](#)

Trees of CTreeNode class descendants - descendants of class CTree - have practical application.

Descendant of CTree class should have a predefined method [CreateElement](#) that creates a new instance of the [CTreeNode](#) descendant class.

Let's consider an example of the CTree descendant class.

```
//+-----+
//|                                     MyTree.mq5 |
//|                                     Copyright 2010, MetaQuotes Software Corp. |
//|                                     https://www.metaquotes.net/ |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\Tree.mqh>
#include "MyTreeNode.mqh"
//---
input int extCountedNodes = 100;
//+-----+
//| Describe class CMyTree derived from CTree. |
//+-----+
//| Class CMyTree. |
//| Purpose: Construction and navigation of a binary search tree. |
//+-----+
class CMyTree : public CTree
{
public:
    //--- methods of searching in the tree by custom data
    CMyTreeNode* FindByLong(long find_long);
    //--- method of creation of the tree element
    virtual CTreeNode *CreateElement();
};
//---
CMyTree MyTree;
//+-----+
//| Creation of a new tree node. |
//| INPUT: none. |
```



```

//| OUTPUT: pointer to the new tree node if OK, or NULL. |
//| REMARK: none. |
//+-----+
CTreeNode *CMyTree::CreateElement()
{
    CMyTreeNode *node=new CMyTreeNode;
//---
    return(node);
}
//+-----+
//| Search of element in a list by value m_long. |
//| INPUT: find_long - searched value. |
//| OUTPUT: pointer of a found list element, or NULL. |
//| REMARK: none. |
//+-----+
CMyTreeNode* CMyTree::FindByLong(long find_long)
{
    CMyTreeNode *res=NULL;
    CMyTreeNode *node;
//--- create a tree node to pass the search parameter
    node=new CMyTreeNode;
    if(node==NULL) return(NULL);
    node.SetLong(find_long);
//---
    res=Find(node);
    delete node;
//---
    return(res);
}
//+-----+
//| script "testing of class CMyTree" |
//+-----+
//--- array for string initialization
string str_array[11]={"p","oo","iii","uuuu","yyyyy","ttttt","rrrr","eee","ww","q","999"};
//---
int OnStart() export
{
    int i;
    uint pos;
    int beg_time,end_time;
    CMyTreeNode *node; //--- temporary pointer to the sample of class CMyTreeNode
//---
    printf("Start test %s.",__FILE__);
//--- Fill out MyTree with instances of class MyTreeNode in the amount of extCountedNodes
    beg_time=GetTickCount();
    for(i=0;i<extCountedNodes;i++)
    {
        node=MyTree.CreateElement();
        if(node==NULL)

```



```

    {
        //--- emergency exit
        printf("%s (%4d): create error",__FILE__,__LINE__);
        return(__LINE__);
    }
    NodeSetData(node,i);
    node.SetLong(i);
    MyTree.Insert(node);
}
end_time=GetTickCount();
printf("Filling time of MyTree is %d ms.",end_time-beg_time);
//--- Create a temporary tree TmpMyTree.
CMyTree TmpMyTree;
//--- Detach 50% of tree elements (all even)
//--- and add them to the temporary tree TmpMyTree.
beg_time=GetTickCount();
for(i=0;i<extCountedNodes;i+=2)
{
    node=MyTree.FindByLong(i);
    if(node!=NULL)
        if(MyTree.Detach(node)) TmpMyTree.Insert(node);
}
end_time=GetTickCount();
printf("Deletion time of %d elements from MyTree is %d ms.",extCountedNodes/2,end_t
//--- Return the detached
node=TmpMyTree.Root();
while(node!=NULL)
{
    if(TmpMyTree.Detach(node)) MyTree.Insert(node);
    node=TmpMyTree.Root();
}
//--- Check work of method Save(int file_handle);
int file_handle;
file_handle=FileOpen("MyTree.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
if(file_handle>=0)
{
    if(!MyTree.Save(file_handle))
    {
        //--- error writing to a file
        //--- emergency exit
        printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
        //--- close file before leaving!!!
        FileClose(file_handle);
        return(__LINE__);
    }
    FileClose(file_handle);
}
//--- Check work of method Load(int file_handle);
file_handle=FileOpen("MyTree.bin",FILE_READ|FILE_BIN|FILE_ANSI);

```



```

    if(file_handle>=0)
    {
        if(!TmpMyTree.Load(file_handle))
        {
            //--- error reading from file
            //--- emergency exit
            printf("%s: Error %d in %d!",__FILE__,__LINE__);
            //--- close file before leaving!!!
            FileClose(file_handle);
            return(__LINE__);
        }
        FileClose(file_handle);
    }
//---
    MyTree.Clear();
    TmpMyTree.Clear();
//---
    printf("End test %s. OK!",__FILE__);
//---
    return(0);
}
//+-----+
//| Function to output node contents to journal |
//+-----+
void NodeToLog(CMyTreeNode *node)
{
    printf("    %I64d,%f,'%s','%s',
           node.GetLong(),node.GetDouble(),
           node.GetString(),TimeToString(node.GetDateTime()));
}
//+-----+
//| Function to populate node with random values |
//+-----+
void NodeSetData(CMyTreeNode *node,int mode)
{
    if(mode%2==0)
    {
        node.SetLong(mode*MathRand());
        node.SetDouble(MathPow(2.02,mode)*MathRand());
    }
    else
    {
        node.SetLong(mode*(long)(-1)*MathRand());
        node.SetDouble(-MathPow(2.02,mode)*MathRand());
    }
    node.SetString(str_array[mode%10]);
    node.SetDateTime(10000*mode);
}

```


Root

Gets the root node of the tree.

```
CTreeNode* Root() const
```

Return Value

Pointer of the root node of the tree.

CreateElement

Creates a new instance of the node.

```
virtual CTreeNode* CreateElement()
```

Return Value

Pointer of the new instance of the node or NULL.

Insert

Adds a node to a tree.

```
CTreeNode* Insert(  
    CTreeNode* new_node    // node  
)
```

Parameters

new_node

[in] Pointer of a node to insert to a tree.

Return Value

Pointer of the owner node or NULL.

Detach

Detaches a specified node from a tree.

```
bool Detach(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] Node pointer to detach.

Return Value

true - success, otherwise false.

Note

After detachment, the node pointer is not released. The tree is balanced.

Delete

Deletes a specified node from a tree.

```
bool Delete(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] Node pointer to delete.

Return Value

true - success, otherwise false.

Note

After deletion, a node pointer is released. The tree is balanced.

Clear

Deletes all nodes of a tree.

```
void Clear()
```

Return Value

None.

Note

After deletion, node pointers are released.

Find

Searches for a node in a tree by sample.

```
CTreeNode* Find(  
    CTreeNode* node    // node  
)
```

Parameters

node

[in] Node that contains data used as a search sample.

Return Value

Pointer of the found node or NULL.

Save

Writes tree data to a file.

```
virtual bool Save(  
    int file_handle    // handle  
)
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true - success, otherwise false.

Load

Reads tree data from a file.

```
virtual bool Load(  
    int file_handle    // handle  
)
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

Return Value

true - success, otherwise false.

Type

Gets identifier of the tree type.

```
virtual int Type() const
```

Return Value

Identifier of the tree type.

File Operations

This section contains the technical details of the file operations classes and descriptions of the corresponding components of the MQL5 standard library.

The file operations classes will save time in developing applications using file input/output operations.

The MQL5 Standard Library (in terms of file operations) is located in the working directory of the terminal in the Include\Files folder.

Class	Description
<u>CFile</u>	Base file operations class
<u>CFileBin</u>	Binary file operations class
<u>CFileTxt</u>	Text file operations class

CFile

CFile is a base class for CFileBin and CFileTxt classes.

Description

CFile class provides the simplified access to MQL5 API file and folder functions for all of its descendants.

Declaration

```
class CFile: public CObject
```

Title

```
#include <Files\File.mqh>
```

Inheritance hierarchy

CObject

CFile

Direct descendants

CFileBin, CFilePipe, CFileTxt

Class Methods by Groups

Attributes	
<u>Handle</u>	Gets file handle
<u>Filename</u>	Gets file name
<u>Flags</u>	Gets file flags
<u>SetUnicode</u>	Sets/clears the FILE_UNICODE flag
<u>SetCommon</u>	Sets/clears the FILE_COMMON flag
General methods for files	
<u>Open</u>	Opens file
<u>Close</u>	Closes file
<u>Delete</u>	Deletes file
<u>IsExist</u>	Checks file for existence
<u>Copy</u>	Copies file
<u>Move</u>	Renames/moves file
<u>Size</u>	Gets file size
<u>Tell</u>	Gets current file position

Seek	Sets current file position
Flush	Flushes data on disk
IsEnding	Checks file for end
IsLineEnding	Checks line for end
General methods for folders	
FolderCreate	Creates folder
FolderDelete	Deletes folder
FolderClean	Clears folder
Files and folders search methods	
FileFindFirst	Begins file search
FileFindNext	Continues file search
FileFindClose	Closes search handle

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Handle

Gets file handle of the opened file.

```
int Handle()
```

Return Value

Handle of the opened file assigned to the class instance. If there is no file assigned, it returns -1.

FileName

Gets file name of the opened file.

```
string FileName()
```

Return Value

File name of the opened fil, assigned to the class instance. If there is no file assigned, it returns "".

Flags

Gets flags of the opened file.

```
int  Flags ()
```

Return Value

Flags of the opened file assigned to the class instance.

SetUnicode

Sets/clears the FILE_UNICODE flag.

```
void SetUnicode(  
    bool unicode    // flag value  
)
```

Parameters

unicode

[in] New value for FILE_UNICODE flag.

Note

The result of string operations is dependent on the FILE_UNICODE flag. If it is false, the ANSI codes are used (one byte symbols). If it set, the UNICODE codes are used (two byte symbols). If the file is already opened, the flag cannot be changed.

SetCommon

Sets/clears the FILE_COMMON flag.

```
void SetCommon(  
    bool common    // flag value  
)
```

Parameters

common

[in] New value for FILE_COMMON flag.

Note

The FILE_COMMON flag determines the current working folder. If it is false, the local terminal folder is used as the current working folder. If it is true, the common data folder is used as the current working folder. If the file is already opened, the flag cannot be changed.

Open

Opens the specified file and, if it successful, assigns it to the class instance.

```
int Open(  
    const string file_name,      // file name  
    int flags,                  // flags  
    short delimiter=9           // separator  
)
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags.

delimiter=9

[in] CSV file separator.

Return Value

Handle of the opened file.

Note

The working folder is dependent on flag that was previously set/reset using the SetCommon() method.

Close

Closes file assigned to the class instance.

```
void Close()
```


Delete

Deletes the file assigned to the file instance.

```
void Delete()
```

Delete

Deletes the specified file.

```
void Delete(  
    const string file_name    // file name  
)
```

Parameters

file_name

[in] File name of the file to delete.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

IsExist

Checks file for existence

```
bool IsExist(  
    const string file_name    // file name  
)
```

Parameters

file_name

[in] Name of the file to check.

Return Value

true - file exists.

Copy

Copies a file.

```
bool Copy(  
    const string src_name,      // file name  
    int src_flag,              // flag  
    const string dst_name,      // file name  
    int dst_flags              // flags  
)
```

Parameters

src_name

[in] Name of a source file.

src_flag

[in] Flags of a source file (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Return Value

true - successful, false - cannot copy the file.

Move

Renames/moves file.

```
bool Move(  
    const string src_name,      // file name  
    int src_flag,              // flag  
    const string dst_name,      // file name  
    int dst_flags              // flags  
)
```

Parameters

src_name

[in] Source file name.

src_flag

[in] Source file flags (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Return Value

true - successful, false - failed to move/rename the file.

Size

Gets file size in bytes.

```
ulong Size()
```

Return Value

file size in bytes. If there is no file assigned, it returns ULONG_MAX.

Tell

Gets the current file pointer's position.

```
ulong Tell()
```

Return Value

the current file position. If there no file assigned, it returns ULONG_MAX.

Seek

Sets file pointer's position.

```
void Seek(  
    long          offset,      // offset  
    ENUM_FILE_POSITION origin  // origin  
)
```

Parameters

offset

[in] File offset in bytes (can be negative).

origin

[in] Origin of the offset.

Return Value

true - successful, false - cannot change the file pointer.

Flush

Flushes all of the file input/output buffer data on disk.

```
void Flush()
```


IsEnding

Checks file for end during the file read operations.

```
bool IsEnding()
```

Return Value

true - end of file has been achieved during read or seek operation.

IsLineEnding

Checks a text file for end during the file read operations.

```
bool IsLineEnding()
```

Return Value

true - end of line has been achieved during a txt or csv file read operation (CR-LF chars).

FolderCreate

Creates new folder.

```
bool FolderCreate(  
    const string folder_name    // folder name  
)
```

Parameters

folder_name

[in] Name of the folder to create. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, false - cannot create the folder.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

FolderDelete

Deletes specified folder.

```
bool FolderDelete(  
    const string folder_name    // folder name  
)
```

Parameters

folder_name

[in] Name of the folder to delete. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, false - cannot delete the folder.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

FolderClean

Cleans specified folder.

```
bool FolderClean(  
    const string folder_name    // folder name  
)
```

Parameters

folder_name

[in] Name of the folder to clean. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, and false - cannot change the folder.

Note

The working folder is dependent on the flag previously set/reset by SetCommon() method.

FileFindFirst

Begins file search using the specified filter.

```
int FileFindFirst(  
    const string filter,          // search filter  
    string&      file_name       // reference  
)
```

Parameters

filter

[in] Search filter.

file_name

[out] The reference to the string the name of the first found file is placed into in case of success.

Return Value

The handle that can be used for further file search using FileFindNext, or INVALID_HANDLE if there are no files corresponding to the filter.

Note

The working folder is dependent on the flag previously set/reset by SetCommon() method.

FileFindNext

Continues file search started by the FileFindFirst() method.

```
bool FileFindNext(  
    int      search_handle,    // search handle  
    string&  file_name        // reference  
)
```

Parameters

search_handle

[in] Search handle returned by FileFindFirst() method.

file_name

[in] The reference to the string the name of the found file is placed into if successful.

Return Value

true - successful, false - there are no files corresponding to the filter.

FileFindClose

Closes search handle.

```
void FileFindClose(  
    int search_handle    // search handle  
)
```

Parameters

search_handle

[in] Search handle returned by FileFindFirst() method.

CFileBin

CFileBin is a class for simplified access to binary files.

Description

CFileBin class provides access to binary files.

Declaration

```
class CFileBin: public CFile
```

Title

```
#include <Files\FileBin.mqh>
```

Inheritance hierarchy

[CObject](#)

[CFile](#)

CFileBin

Class Methods by Groups

Open methods	
Open	Opens a binary file
Write methods	
WriteChar	Writes char or uchar type variable
WriteShort	Writes short or ushort type variable
WriteInteger	Writes int or uint type variable
WriteLong	Writes long or ulong type variable
WriteFloat	Writes float type variable
WriteDouble	Writes double type variable
WriteString	Writes string type variable
WriteCharArray	Writes an array of char or uchar type variables
WriteShortArray	Writes an array of short or ushort type variables
WriteIntegerArray	Writes an array of int or uint type variables
WriteLongArray	Writes an array of long or ulong type variables
WriteFloatArray	Writes an array of float variables
WriteDoubleArray	Writes an array of double type variables

WriteObject	Writes data of the CObject class inheritor instance
Read methods	
ReadChar	Reads char or uchar type variable
ReadShort	Reads short or ushort type variable
ReadInteger	Reads int or uint type variable
ReadLong	Reads long or ulong type variable
ReadFloat	Reads float type variable
ReadDouble	Reads double type variable
ReadString	Reads string type variable
ReadCharArray	Reads an array of char or uchar type variables
ReadShortArray	Reads an array of short or ushort type variables
ReadIntegerArray	Reads an array of int or uint type variables
ReadLongArray	Reads an array of long or ulong type variables
ReadFloatArray	Reads an array of float type variables
ReadDoubleArray	Reads an array of double type variables
ReadObject	Reads data of the CObject class inheritor instance

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CFile

[Handle](#), [FileName](#), [Flags](#), [SetUnicode](#), [SetCommon](#), [Open](#), [Close](#), [Delete](#), [Size](#), [Tell](#), [Seek](#), [Flush](#), [IsEnding](#), [IsLineEnding](#), [Delete](#), [IsExist](#), [Copy](#), [Move](#), [FolderCreate](#), [FolderDelete](#), [FolderClean](#), [FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

Open

Opens the specified binary file and, if successful, assigns it to the class instance.

```
int Open(  
    const string file_name,    // file name  
    int flags                  // flags  
)
```

Parameters

file_name

[in] File name of the file to open.

flags

[in] File open flags (the FILE_BIN flag is set forcibly).

Return Value

Handle of the opened file.

WriteChar

Writes char or uchar type variable to file.

```
uint WriteChar(  
    char value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteShort

Writes short or ushort type variable to file.

```
uint WriteShort(  
    short value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteInteger

Writes int or uint type variable to file.

```
uint WriteInteger(  
    int value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteLong

Writes long or ulong type variable to file.

```
uint WriteLong(  
    long value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteFloat

Writes float type variable to file.

```
uint WriteFloat(  
    float value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteDouble

Writes double type variable to file.

```
uint WriteDouble(  
    double value    // value  
)
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint WriteString(  
    const string value    // value  
)
```

Parameters

value

[in] String to write.

Return Value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint WriteString(  
    const string value,    // value  
    int size              // size  
)
```

Parameters

value

[in] String to write.

size

[in] Number of bytes to write.

Return Value

Number of bytes written.

WriteCharArray

Writes an array of char or uchar type variables to file.

```
uint WriteCharArray(  
    char& array[],           // array  
    int start_item=0,        // start element  
    int items_count=-1       // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteShortArray

Writes an array of short or ushort type variables to file.

```
uint WriteShortArray(  
    short& array[],           // array  
    int start_item=0,         // start element  
    int items_count=-1        // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteIntegerArray

Writes an array of int or uint type variables to file.

```
uint WriteIntegerArray(  
    int& array[],           // array  
    int start_item=0,       // start element  
    int items_count=-1      // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteLongArray

Writes an array of long or ulong type variables to file.

```
uint WriteLongArray(  
    long& array[],           // array  
    int start_item=0,        // start element  
    int items_count=-1       // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteFloatArray

Writes an array of float type variables to file.

```
uint WriteFloatArray(  
    float& array[],           // array  
    int    start_item=0,      // start element  
    int    items_count=-1     // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteDoubleArray

Writes an array of double type variables to file.

```
uint WriteDoubleArray(  
    double& array[],           // array to write  
    int start_item=0,         // start element  
    int items_count=-1        // number of elements  
)
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteObject

Writes data of the CObject class inheritor instance to file.

```
bool WriteObject(  
    CObject* object    // reference to the object  
)
```

Parameters

object

[in] Reference to the CObject class inheritor instance to write.

Return Value

true - successful, false - cannot write the data.

ReadChar

Reads char or uchar type variable from file.

```
bool ReadChar(  
    char& value    // flag value  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadShort

Reads short or ushort type variable from file.

```
bool ReadShort(  
    short& value  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadInteger

Reads int or uint type variable from file.

```
bool ReadInteger(  
    int& value    // variable  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadLong

Reads long or ulong type variable from file.

```
bool ReadLong(  
    long& value  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadFloat

Reads float type variable from file.

```
bool ReadFloat(  
    float& value    // variable  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadDouble

Reads double type variable from file.

```
bool ReadDouble(  
    double& value  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadString

Reads string type variable from file.

```
bool ReadString(  
    string& value    // string  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadString

Reads string type variable from file.

```
bool ReadString(  
    string& value  
)
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadCharArray

Reads an array of char or uchar type variables from file.

```
bool ReadCharArray(  
    char& array[],           // array  
    int start_item=0,        // start element  
    int items_count=-1       // number of elements  
)
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadShortArray

Reads an array of short or ushort type variables from file.

```
bool ReadShortArray(  
    short& array[],           // array  
    int start_item=0,        // start element  
    int items_count=-1       // number of elements  
)
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadIntegerArray

Reads an array of int or uint type variables from file.

```
bool ReadIntegerArray(  
    int& array[],           // array  
    int start_item=0,      // start element  
    int items_count=-1     // number of elements  
)
```

Parameters

array[]

[in] Reference to the target array of type int or uint.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadLongArray

Reads an array of long or ulong type variables from file.

```
bool ReadLongArray(  
    long& array[],           // array  
    int start_item=0,       // start element  
    int items_count=-1      // number of elements  
)
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadFloatArray

Reads an array of float type variables from file.

```
bool ReadFloatArray(  
    float& array[],           // array  
    int start_item=0,        // start element  
    int items_count=-1       // number of elements  
)
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadDoubleArray

Reads an array of double type variables from file.

```
bool ReadDoubleArray(  
    double& array[],           // array  
    int start_item=0,         // start element  
    int items_count=-1        // number of elements  
)
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadObject

Reads data of the CObject class inheritor instance from file.

```
bool ReadObject(  
    CObject* object    // pointer  
)
```

Parameters

object

[in] Pointer to the CObject class inheritor instance to read.

Return Value

true - successful, false - cannot read the data.

CFileTxt

CFileTxt is a class for simplified access to text files.

Description

CFileTxt class provides access to text files.

Declaration

```
class CFileTxt: public CFile
```

Title

```
#include <Files\FileTxt.mqh>
```

Inheritance hierarchy

[CObject](#)

[CFile](#)

CFileTxt

Class Methods by Groups

Open methods	
Open	Opens a text file
Write methods	
WriteString	Writes string type variable
Read methods	
ReadString	Reads string type variable

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CFile

[Handle](#), [FileName](#), [Flags](#), [SetUnicode](#), [SetCommon](#), [Open](#), [Close](#), [Delete](#), [Size](#), [Tell](#), [Seek](#), [Flush](#), [IsEnding](#), [IsLineEnding](#), [Delete](#), [IsExist](#), [Copy](#), [Move](#), [FolderCreate](#), [FolderDelete](#), [FolderClean](#), [FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

Open

Opens the specified text file and, if successful, assigns it to the class instance.

```
int Open(  
    const string file_name,    // file name  
    int flags                 // flags  
)
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags (FILE_TXT flag is forcibly set).

Return Value

Opened file handle.

WriteString

Writes string type variable to file.

```
uint WriteString(  
    const string value    // string  
)
```

Parameters

value

[in] String to write.

Return Value

Number of bytes written.

ReadString

Reads string type variable from file.

```
string ReadString()
```

Return Value

String which has been read.

String operations

This section contains the technical details of the string operations classes and descriptions of the corresponding components of the MQL5 standard library.

The use of string operations classes will save time in developing applications processing textual data.

The MQL5 standard library (in terms of string operations) is located in the working directory of the terminal in the Include\Strings folder.

Class	Description
<u>CString</u>	Class for string operations

CString

CString is a class for simplified access to the variables of string type.

Description

CString class provides simplified access to MQL5 API functions working with string variables.

Declaration

```
class CString: public CObject
```

Title

```
#include <Strings\String.mqh>
```

Inheritance hierarchy

[CObject](#)

CString

Class Methods by Groups

Data access methods	
Str	Gets a string
Len	Gets length of a string
Copy	Copies a string copy
Fill methods	
Fill	Fills a string
Assign	Assigns a string
Append	Appends a string
Insert	Inserts a string
Compare methods	
Compare	Compares strings
CompareNoCase	Performs a case insensitive string comparison
Substring methods	
Left	Gets a substring from the left
Right	Gets a substring from the right
Mid	Gets a substring from the middle
Trim/delete methods	

Trim	Trims a string from the left and from the right
TrimLeft	Trims a string from the left
TrimRight	Trims a string from the right
Clear	Clears a string
Convert methods	
ToUpper	Converts a string to uppercase.
ToLower	Converts a string to lowercase.
Reverse	Reverses a string
Search methods	
Find	Searches a substring left to right
FindRev	Searches a substring right to left
Remove	Deletes a substring
Replace	Replaces a substring

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#)

Str

Gets a string.

```
string Str() const;
```

Return Value

Copy of a string.

Len

Gets length of a string.

```
uint Len() const;
```

Return Value

Length of a string.

Copy

Copies a string by reference.

```
void Copy(  
    string& copy      // reference  
    ) const;
```

Parameters

copy

[in] Reference to a data string.

Copy

Copies a string to the CString class instance.

```
void Copy(  
    CString* copy      // pointer  
    ) const;
```

Parameters

copy

[in] Pointer to CString class instance for data.

Fill

Fills a string with specified char.

```
bool Fill(  
    short character    // character  
)
```

Parameters

character

[in] Character for filling a string.

Return Value

true - successful, false - cannot fill a string.

Assign

Assigns a string.

```
void Assign(  
    const string str    // string  
)
```

Parameters

str

[in] String to assign.

Assign

Assigns a string from the CString class instance.

```
void Assign(  
    CString* str        // pointer  
)
```

Parameters

str

[in] Pointer to the CString class instance to assign.

Append

Appends a string.

```
void Append(  
    const string str      // string  
)
```

Parameters

str

[in] String to append.

Append

Appends a string from the CString class instance.

```
void Append(  
    CString* string      // pointer  
)
```

Parameters

string

[in] Pointer to the CString class instance to append.

Insert

Inserts a string to the specified position.

```
uint Insert(  
    uint      pos,      // position  
    const string str     // string  
)
```

Parameters

pos

[in] Insert position.

str

[in] String to insert.

Return Value

Resulted string length.

Insert

Inserts a string to the specified position from the CString class instance.

```
uint Insert(  
    uint      pos,      // position  
    CString*  str        // pointer  
)
```

Parameters

pos

[in] Position to insert into.

str

[in] Pointer to the CString class instance to insert.

Return Value

Resulted string length.

Compare

Compares to a string.

```
int Compare(  
    const string str    // string  
    ) const;
```

Parameters

str

[in] String to compare.

Return Value

0 - both strings are equal, -1 - the class string is lower than the string to compare, 1 - the class string is greater than the string to compare.

Compare

Compares to a CString class instance string.

```
int Compare(  
    CString* str        // pointer  
    ) const;
```

Parameters

str

[in] Pointer to CString class instance to compare.

Return Value

0 - strings are equal, -1 - the class string is lower than the string to compare, 1 - the class string is greater than the string to compare.

CompareNoCase

Performs a case insensitive string comparison.

```
int CompareNoCase(  
    const string str      // string  
    ) const;
```

Parameters

str

[in] String to compare.

Return Value

0 - strings are equal, -1 - a class string is lower than a string to compare, 1 - a class string is greater than a string to compare.

CompareNoCase

Compares a string (case insensitive) to a CString class instance string.

```
int CompareNoCase(  
    CString* str          // pointer  
    ) const;
```

Parameters

str

[in] Pointer to CString class instance to compare.

Return Value

0 - if strings are equal, -1 - a class string is lower than a string to compare, 1 - a class string is greater than a string to compare.

Left

Gets a substring of a specified length from the beginning of a string.

```
string Left(  
    uint count    // length  
)
```

Parameters

count

[in] Substring length.

Return Value

Resulted substring.

Right

Gets a substring of a specified length from the end of a string.

```
string Right(  
    uint count    // length  
)
```

Parameters

count

[in] Substring length.

Return Value

Resulted substring.

Mid

Gets a substring of a specified length from a specified string position.

```
string Mid(  
    uint pos,           // position  
    uint count          // length  
)
```

Parameters

pos

[in] Substring position.

count

[in] Substring length.

Return Value

Resulted substring.

Trim

Removes all characters within a set (as well as ' ', '\t', '\r', '\n') at both ends of a string from this string.

```
int Trim(  
    const string targets    // set  
)
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of removed characters.

Example:

```
///--- example for CString::Trim  
#include <Strings\String.mqh>  
///---  
void OnStart()  
{  
    CString str;  
    ///---  
    str.Assign(" \t\tABCD\r\n");  
    printf("Source string '%s'", str.Str());  
    ///---  
    str.Trim("DA-DA-DA");  
    printf("Result string '%s'", str.Str());  
}
```


TrimLeft

Removes all characters within a set (as well as ' ', '\t', '\r', '\n') at the beginning of a string from this string.

```
int TrimLeft(  
    const string targets    // set  
)
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of characters removed.

TrimRight

Removes all characters within a set (as well as ' ', '\t', '\r', '\n') at the end of a string from this string.

```
int TrimRight(  
    const string targets    // set  
)
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of characters removed.

Clear

Clears a string.

```
bool Clear()
```

Return Value

true - successful, false - cannot clear a string.

ToUpper

Converts all string characters to uppercase.

```
bool ToUpper ()
```

Return Value

true - successful, false - cannot convert to uppercase.

ToLower

Converts all string characters to lowercase.

```
bool ToLower ()
```

Return Value

true - successful, false - cannot convert to lowercase.

Reverse

Reverses a string (initial and final characters exchange places pair-wise).

```
void Reverse ()
```


Find

Searches for the first match of a substring from a specified position.

```
int Find(  
    uint      start,          // position  
    const string substring    // substring  
) const;
```

Parameters

start

[in] Initial position for substring search.

substring

[in] Sample substring to search for.

Return Value

The index of the first match of a substring (-1 - substring is not found).

FindRev

Searches for the last match of a substring.

```
int FindRev(  
    const string substring    // substring  
    ) const;
```

Parameters

substring

[in] Sample substring to search for.

Return Value

The index of the last match of a substring (-1 - substring is not found).

Remove

Removes all substring matches.

```
uint Remove(  
    const string substring // substring  
)
```

Parameters

substring

[in] Sample substring to search for.

Return Value

Number of substring removals.

Replace

Replaces all substring matches.

```
uint Replace(  
    const string  substring,    // substring  
    const string  newstring     // substring  
)
```

Parameters

substring

[in] Sample substring to search for.

newstring

[in] Sample substring to replace with.

Return Value

Number of substring replacements.

Graphic Objects

This section contains the technical details of working with classes of graphical objects and a description of the relevant components of the MQL5 Standard Library.

The use of classes of graphical objects will save time when creating custom programs (scripts, Expert Advisors).

MQL5 Standard Library (in terms of graphical objects) is located in the working directory of the terminal in the Include\ChartObjects folder.

Class/Group	Description
<u>CChartObject</u>	Base class of a graphic object
<u>Lines</u>	Group classes "Lines"
<u>Channels</u>	Group classes "Channels"
<u>Gann Tools</u>	Group classes "Gann"
<u>Fibonacci Tools</u>	Group classes "Fibonacci"
<u>Elliott Tools</u>	Group classes "Elliott"
<u>Shapes</u>	Group classes "Shapes"
<u>Arrows</u>	Group classes "Arrows"
<u>Controls</u>	Group classes "Controls"

CChartObject

CChartObject is a base class for the classes of chart graphical objects of the Standard MQL5 library.

Description

CChartObject class provides the simplified access to MQL5 API functions for all of its descendants.

Declaration

```
class CChartObject : public CObject
```

Title

```
#include <ChartObjects\ChartObject.mqh>
```

Inheritance hierarchy

[CObject](#)

CChartObject

Direct descendants

[CChartObjectArrow](#), [CChartObjectBitmap](#), [CChartObjectBmpLabel](#), [CChartObjectCycles](#),
[CChartObjectElliottWave3](#), [CChartObjectEllipse](#), [CChartObjectFiboArc](#), [CChartObjectFiboFan](#),
[CChartObjectFiboTimes](#), [CChartObjectHLine](#), [CChartObjectRectangle](#), [CChartObjectSubChart](#),
[CChartObjectText](#), [CChartObjectTrend](#), [CChartObjectTriangle](#), [CChartObjectVLine](#)

Class Methods by Groups

Attributes	
ChartId	Gets the ID of the chart a graphical object belongs to
Window	Gets the number of a chart window where a graphical object is located
Name	Gets/sets the name of a graphical object
NumPoints	Gets the number of anchor points
Assign	
Attach	Binds a chart graphical object
SetPoint	Sets the anchor point parameters
Delete	
Delete	Deletes a chart graphical object
Detach	Detaches a chart graphical object
Shift	

<u>ShiftObject</u>	The relative object shift
<u>ShiftPoint</u>	The relative object point shift
Object properties	
<u>Time</u>	Gets/sets the time coordinates of an object point
<u>Price</u>	Gets/sets the price coordinate of an object point
<u>Color</u>	Gets/sets the color of the object
<u>Style</u>	Gets/sets the object line style
<u>Width</u>	Gets/sets the object line width
<u>BackGround</u>	Gets/sets the flag of drawing an object in the background
<u>Selected</u>	Gets/sets the "selected" flag of a graphical object
<u>Selectable</u>	Gets/sets the selectable object flag
<u>Description</u>	Gets/sets the text of the object
<u>Tooltip</u>	Gets/sets the tooltip of the object
<u>Timeframes</u>	Gets/sets the mask of the object visibility flags
<u>Z_Order</u>	Gets/sets the graphical object priority for receiving an event of mouse clicking on a chart
<u>CreateTime</u>	Gets the time of the object creation
Object level properties	
<u>LevelsCount</u>	Gets/sets the number of object levels
<u>LevelColor</u>	Gets/sets the level line color
<u>LevelStyle</u>	Gets/sets the level line style
<u>LevelWidth</u>	Gets/sets the level line width
<u>LevelValue</u>	Gets/sets the level
<u>LevelDescription</u>	Gets/sets the level text
Access to API MQL5 functions	
<u>GetInteger</u>	Gets the value of the object property
<u>SetInteger</u>	Sets the value of the object property
<u>GetDouble</u>	Gets the value of the object property
<u>SetDouble</u>	Sets the value of the object property

GetString	Gets the value of the object property
SetString	Sets the value of the object property
Input/Output	
virtual Save	Virtual method of writing to a file
virtual Load	Virtual method of reading from a file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

ChartId

Gets the ID of the chart a graphical object belongs to.

```
long ChartId() const
```

Return Value

ID of the chart where the graphical object is located. If there is no bound object, it returns -1.

Example:

```
//--- example for CChartObject::ChartId
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get chart identifier of chart object
    long chart_id=object.ChartId();
}
```


Window

Gets the index of the chart window where the graphical object is located.

```
int Window() const
```

Return Value

The number of the chart window where the graphical object is located (0 - main window). If there is no bound object, it returns -1.

Example:

```
//--- example for CChartObject::Window
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get window of chart object
    int window=object.Window();
}
```


Name (Get Method)

Gets the name of the graphical object.

```
string Name() const
```

Return Value

Name of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Name (Set Method)

Sets the name of the graphical object.

```
bool Name (  
    string name      // new name  
)
```

Parameters

name

[in] The new name of the graphical object.

Return Value

true - success, false - cannot change the name.

Example:

```
//--- example for CChartObject::Name  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //--- get name of chart object  
    string object_name=object.Name();  
    if(object_name!="MyChartObject")  
    {  
        //--- set name of chart object  
        object.Name("MyChartObject");  
    }  
}
```


NumPoints

Gets the number of anchor points of a graphical object.

```
int NumPoints() const
```

Return Value

Number of points linking a graphical object attached to an instance of the class. If there is no attached object, it returns 0.

Example:

```
//--- example for CChartObject::NumPoints
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get points count of chart object
    int points=object.NumPoints();
}
```


Attach

Attaches a graphical object to an instance of the class.

```
bool Attach(  
    long    chart_id,      // chart ID  
    string  name,          // name of the object  
    int     window,        // chart window  
    int     points         // number of points  
)
```

Parameters

chart_id

[out] Chart identifier.

name

[in] Name of the graphical object.

window

[in] Chart window number (0 - main window).

points

[in] Number of anchor points of the graphical object.

Return Value

true - success, false - cannot bind the object.

Example:

```
///--- example for CChartObject::Attach  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- attach chart object  
    if(!object.Attach(CharID(), "MyObject", 0, 2))  
    {  
        printf("Object attach error");  
        return;  
    }  
}
```


SetPoint

Sets new coordinates of the specified anchor point of the graphical object.

```
bool SetPoint(  
    int      point,          // point number  
    datetime new_time,       // time coordinate  
    double   new_price       // price coordinate  
)
```

Parameters

point

[in] Number of the graphical object anchor point.

new_time

[in] New value for the time coordinate of the specified anchor point.

new_price

[in] New value for price coordinate of the specified anchor point.

Return Value

true - success, false - cannot change coordinates of the point.

Example:

```
///--- example for CChartObject::SetPoint  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    double       price;  
    ///---  
    if(object.NumPoints()>0)  
    {  
        ///--- set point of chart object  
        object.SetPoint(0,CurrTime(),price);  
    }  
}
```


Delete

Removes an attached graphical object from the chart.

```
bool Delete()
```

Return Value

true - success, false - cannot remove the object.

Example:

```
//--- example for CChartObject::Delete
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- detach chart object
    if(!object.Delete())
    {
        printf("Object delete error");
        return;
    }
}
```


Detach

Detaches the graphical object.

```
void Detach()
```

Return Value

None.

Example:

```
//--- example for CChartObject::Detach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- detach chart object
    object.Detach();
}
```


ShiftObject

Shifts a graphical object.

```
bool ShiftObject(  
    datetime d_time,      // increment of time coordinate  
    double   d_price      // increment of price coordinate  
)
```

Parameters

d_time

[in] Increment of the time coordinate of all anchor points.

d_price

[in] Increment of the price coordinate of all anchor points.

Return Value

true - success, false - cannot shift the object.

Example:

```
//--- example for CChartObject::ShiftObject  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    datetime   d_time;  
    double     d_price;  
    //--- shift chart object  
    object.ShiftObject(d_time,d_price);  
}
```


ShiftPoint

Shifts a specified anchor point of the graphical object.

```
bool ShiftPoint(  
    int      point,          // point number  
    datetime d_time,        // increment of time coordinate  
    double   d_price         // increment of price coordinate  
)
```

Parameters

point

[in] Number of a graphical object anchor point.

d_time

[in] Increment of the time coordinate of the specified point.

d_price

[in] Increment of the price coordinate of the specified point.

Return Value

true - success, false - cannot shift the point.

Example:

```
///--- example for CChartObject::ShiftPoint  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    datetime      d_time;  
    double        d_price;  
    ///---  
    if(object.NumPoints()>0)  
    {  
        ///--- shift point of chart object  
        object.ShiftPoint(0,d_time,d_price);  
    }  
}
```


Time (Get Method)

Gets the time coordinate of the specified anchor point of a graphical object.

```
datetime Time(  
    int point      // point number  
) const
```

Parameters

point

[in] Number of a graphical object anchor point.

Return Value

Time coordinate of the specified anchor point of the graphical object attached to an instance of the class. If there is no attached object or the object does not have this point, it returns 0.

Time (Set Method)

Sets the time coordinate of the specified anchor point of a graphical object.

```
bool Time(  
    int point,      // point number  
    datetime new_time // time  
)
```

Parameters

point

[in] Number of a graphical object anchor point.

new_time

[in] New value for the time coordinate of the specified graphical object anchor point.

Return Value

true - success, false - cannot change the time coordinate.

Example:

```
///--- example for CChartObject::Time  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///---  
    for(int i=0;i<object.NumPoints();i++)  
    {  
        ///--- get time of the chart object point  
        datetime point_time=object.Time(i);  
        if(point_time==0)
```



```
{  
    //--- set time of the chart object point  
    object.Time(i,TimeCurrent());  
}  
}  
}
```


Price (Get Method)

Gets the price coordinate of the specified anchor point of a graphical object.

```
double Price(  
    int point    // point number  
) const
```

Parameters

point

[in] Number of a graphical object anchor point.

Return Value

Price coordinate of the specified anchor point of the graphical object attached to an instance of the class. If there is no attached object or the object does not have this point, it returns [EMPTY_VALUE](#).

Price (Set Method)

Sets the price coordinate of the specified anchor point of a graphical object.

```
bool Price(  
    int point,      // point number  
    double new_price // price  
)
```

Parameters

point

[in] Number of a graphical object anchor point.

new_price

[in] New value for the price coordinate of the specified graphical object anchor point.

Return Value

true - success, false - cannot change the price coordinate.

Example:

```
///--- example for CChartObject::Price  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    double price;  
    ///---  
    for(int i=0;i<object.NumPoints();i++)  
    {  
        ///--- get price of the chart object point  
        double point_price=object.Price(i);
```



```
if(point_price!=price)
{
    ///--- set price for the chart object point
    object.Price(i,price);
}
}
```


Color (Get Method)

Gets the line color of the graphical object.

```
color Color() const
```

Return Value

Line color of the graphical object attached to the class instance. If there is no object attached, it returns CLR_NONE.

Color (Set Method)

Sets the line color of the graphical object.

```
bool Color(  
    color new_color    // new color  
)
```

Parameters

new_color

[in] New value of a graphical object line color.

Return Value

true - success, false - cannot change the color.

Example:

```
///--- example for CChartObject::Color  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- get color of chart object  
    color object_color=object.Color();  
    if(object_color!=clrRed)  
    {  
        ///--- set color of chart object  
        object.Color(clrRed);  
    }  
}
```


Style (Get Method)

Gets the line style of the graphical object.

```
ENUM_LINE_STYLE Style() const
```

Return Value

Line style of the graphical object attached to the class instance. If there is no attached object, it returns WRONG_VALUE.

Style (Set Method)

Sets the line style of the graphical object.

```
bool Style(  
    ENUM_LINE_STYLE new_style    // style  
)
```

Parameters

new_style

[in] New value of the graphical object line style.

Return Value

true - success, false - cannot change the style.

Example:

```
//--- example for CChartObject::Style  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //--- get style of chart object  
    ENUM_LINE_STYLE style=object.Style();  
    if(style!=STYLE_SOLID)  
    {  
        //--- set style of chart object  
        object.Style(STYLE_SOLID);  
    }  
}
```


Width (Get Method)

Gets the line width of the graphical object.

```
int Width() const
```

Return Value

The line width of the graphical object attached to an instance of the class. If there is no attached object, it returns -1.

Width (Set Method)

Sets the line width of the graphical object.

```
bool Width(  
    int new_width    // thickness  
)
```

Parameters

new_width

[in] New value of the graphical object line width.

Return Value

true - success, false - cannot change the width.

Example:

```
///--- example for CChartObject::Width  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- get width of chart object  
    int width=object.Width();  
    if(width!=1)  
    {  
        ///--- set width of chart object  
        object.Width(1);  
    }  
}
```


Background (Get Method)

Gets the flag for drawing a graphical object on the background.

```
bool Background() const
```

Return Value

Flag for drawing the graphical object, attached to an instance of the class, on the background. If there is no attached object, returns false.

Background (Set Method)

Sets the flag for drawing a graphical object on the background.

```
bool Background(  
    bool background    // value of the flag  
)
```

Parameters

background

[in] New value of the flag for drawing a graphical object on the background.

Return Value

true - success, false - cannot change the flag.

Example:

```
///--- example for CChartObject::Background  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- get background flag of chart object  
    bool background_flag=object.Background();  
    if(!background_flag)  
    {  
        ///--- set background flag of chart object  
        object.Background(true);  
    }  
}
```


Selected (Get Method)

Gets the flag indicating that a graphical object is selected. In other words - if the graphical object is selected or not.

```
bool Selected() const
```

Return Value

The state that the object, attached to an instance of the class, is selected. If there is no attached object, returns false.

Selected (Set Method)

Sets the flag indicating that the graphical object is selected.

```
bool Selected(  
    bool selected    // value of the flag  
)
```

Parameters

selected

[in] New value of the flag indicating that a graphical object is selected.

Return Value

true - success, false - cannot change the flag.

Example:

```
//--- example for CChartObject::Selected  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //--- get the "selected" flag of chart object  
    bool selected_flag=object.Selected();  
    if(selected_flag)  
    {  
        //--- set the "selected" flag of chart object  
        object.Selected(false);  
    }  
}
```


Selectable (Get Method)

Gets the flag indicating an ability of a graphical object to be selected. In other words - if the graphical object can be selected or not.

```
bool Selectable() const
```

Return Value

Flag indicating the ability of the graphical object, attached to an instance of the class, to be selected. If there is no attached object, returns false.

Selectable (Set Method)

Sets the flag indicating the ability of a graphical object to be selected.

```
bool Selectable(  
    bool selectable    // value of the flag  
)
```

Parameters

selectable

[in] New value of the flag indicating an ability of a graphical object to be selected.

Return Value

true - success, false - cannot change the flag.

Example:

```
//--- example for CChartObject::Selectable  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //--- get the "selectable" flag of chart object  
    bool selectable_flag=object.Selectable();  
    if(selectable_flag)  
    {  
        //--- set the "selectable" flag of chart object  
        object.Selectable(false);  
    }  
}
```


Description (Get Method)

Gets a description (text) of a graphical object.

```
string Description() const
```

Return Value

Description (text) of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Description (Set Method)

Sets the description (text) of the graphical object.

```
bool Description(  
    string text    // text  
)
```

Parameters

text

[in] New description (text) of a graphical object.

Return Value

true - success, false - cannot change the description (text).

Example:

```
///--- example for CChartObject::Description  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- get description of chart object  
    string description=object.Description();  
    if(description=="")  
    {  
        ///--- set description of chart object  
        object.Description("MyObject");  
    }  
}
```


Tooltip (Get Method)

Gets the tooltip text of a graphical object.

```
string Tooltip() const
```

Return Value

The text of a tooltip of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Tooltip (Set Method)

Sets the text of the tooltip of a graphical object.

```
bool Tooltip(  
    string new_tooltip    // new text of a tooltip  
)
```

Parameters

new_tooltip

[in] New text of a graphical object tooltip.

Return Value

true - success, false - cannot change the tooltip.

Note:

If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value.

Timeframes (Get Method)

Gets visibility flags of a graphical object.

```
int Timeframes() const
```

Return Value

Visibility flags of the graphical object attached to an instance of the class. If there is no attached object, it returns 0.

Timeframes (Set Method)

Sets visibility flags of a graphical object.

```
bool Timeframes (
    int new_timeframes    // visibility flags
)
```

Parameters

new_timeframes

[in] New visibility flags of the graphical object.

Return Value

true - success, false - cannot change the flags of visibility.

Example:

```
///--- example for CChartObject::Timeframes
#include <ChartObjects\ChartObject.mqh>
///---
void OnStart()
{
    CChartObject object;
    ///--- get timeframes of chart object
    int timeframes=object.Timeframes();
    if(!(timeframes&OBJ_PERIOD_H1))
    {
        ///--- set timeframes of chart object
        object.Timeframes(timeframes|OBJ_PERIOD_H1);
    }
}
```


Z_Order (Get Method)

Gets the graphical object priority for receiving an event of mouse clicking on a chart ([CHARTEVENT_CLICK](#)).

```
long Z_Order() const
```

Return Value

Priority of a graphical object, attached to the class instance. If there is no object attached, it returns 0.

Z_Order (Set Method)

Sets the graphical object priority for receiving an event of mouse clicking on a chart ([CHARTEVENT_CLICK](#)).

```
bool Z_Order(  
    long value    // graphical object priority  
)
```

Parameters

value

[in] New value of priority of a graphical object for receiving the event [CHARTEVENT_CLICK](#).

Return Value

true - success, false - cannot change the priority.

Note

Z_Order property manages a priority when handling clicks on graphical objects. By setting the value greater than 0 (default value), you can increase the object priority when handling mouse clicks.

CreateTime

Gets the graphical object creation time.

```
datetime CreateTime() const
```

Return Value

Creation time of the graphical object attached to the instance of the class. If there is no attached object, it returns 0.

Example:

```
//--- example for CChartObject::CreateTime
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get create time of chart object
    datetime create_time=object.CreateTime();
}
```


LevelsCount (Get Method)

Gets the number of levels of a graphical object.

```
int LevelsCount() const
```

Return Value

Number of levels of the graphical object attached to an instance of the class. If there is no attached object, it returns 0.

LevelsCount (Set Method)

Sets the number of levels of the graphical object.

```
bool LevelsCount(  
    int levels    // number of levels  
)
```

Parameters

levels

[in] The new number of levels of the graphical object.

Return Value

true - success, false - cannot change the number of levels.

Example:

```
//--- example for CChartObject::LevelsCount  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //--- get levels count of chart object  
    int levels_count=object.LevelsCount();  
    //--- set levels count of chart object  
    object.LevelsCount(levels_count+1);  
}
```


LevelColor (Get Method)

Gets the line color of the specified level of a graphical object.

```
color LevelColor(  
    int level    // level number  
) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line color of the specified level of the graphical object attached to the instance of the class. If there is no attached object or the object does not have the specified level, it returns CLR_NONE.

LevelColor (Set Method)

Sets the line color of the specified level of the graphical object.

```
bool LevelColor(  
    int level,        // level number  
    color new_color    // new color  
)
```

Parameters

level

[in] Number of graphical object level.

new_color

[in] New line color of the specified level of a graphical object.

Return Value

true - success, false - cannot change the color.

Example:

```
///--- example for CChartObject::LevelColor  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///---  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        ///--- get level color of chart object  
        color level_color=object.LevelColor(i);  
        if(level_color!=clrRed)
```



```
{  
    //--- set level color of chart object  
    object.LevelColor(i,clrRed);  
}  
}  
}
```


LevelStyle (Get Method)

Gets the line style of the specified level of a graphical object.

```
ENUM_LINE_STYLE LevelStyle(  
    int level    // level number  
) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line style of the specified level of the graphical object attached to an instance of the class. If there is no attached object or the object does not have the specified level, it returns WRONG_VALUE.

LevelStyle (Set Method)

Sets the line style of the specified level of the graphic object.

```
int LevelStyle(  
    int level,    // level number  
    ENUM_LINE_STYLE style // line style  
)
```

Parameters

level

[in] Number of graphical object level.

style

[in] New line style of the specified level of a graphical object.

Return Value

true - success, false - cannot change the style.

Example:

```
//--- example for CChartObject::LevelStyle  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //---  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        //--- get level style of chart object  
        ENUM_LINE_STYLE level_style=object.LevelStyle(i);  
        if(level_style!=STYLE_SOLID)
```



```
{  
    //--- set level style of chart object  
    object.LevelStyle(i,STYLE_SOLID);  
}  
}  
}
```


LevelWidth (Get Method)

Gets the line width of the specified level of a graphical object.

```
int LevelWidth(  
    int level    // level number  
) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line width of the specified level of the graphical object attached to the instance of the class. If there is no attached object or the object does not have the specified level, it returns -1.

LevelWidth (Set Method)

Sets the line width of the specified level of the graphical object.

```
bool LevelWidth(  
    int level,        // level number  
    int new_width     // new width  
)
```

Parameters

level

[in] Number of graphical object level.

new_width

[in] New line width of the specified level of a graphical object.

Return Value

true - success, false - cannot change the width.

Example:

```
//--- example for CChartObject::LevelWidth  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //---  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        //--- get level width of chart object  
        int level_width=object.LevelWidth(i);  
        if(level_width!=1)
```



```
{  
    //--- set level width of chart object  
    object.LevelWidth(i,1);  
}  
}  
}
```


LevelValue (Get Method)

Gets the value of the specified level of a graphical object.

```
double LevelValue(  
    int level    // level number  
) const
```

Parameters

level

[in] Number of a graphical object level.

Return Value

The value of the specified level of a graphical object that is bound to an instance of the class. If there is no bound object or the object has no level specified, returns [EMPTY_VALUE](#).

LevelValue (Set Method)

Sets the value of the specified level of the graphical object.

```
bool LevelValue(  
    int level,        // level number  
    double new_value  // new value  
)
```

Parameters

level

[in] Number of a graphical object level.

new_value

[in] New value of the specified level of a graphical object.

Return Value

true - successful, false - cannot change the value.

Example:

```
//--- example for CChartObject::LevelValue  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    CChartObject object;  
    //---  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        //--- get level value of chart object  
        double level_value=object.LevelValue(i);  
        if(level_value!=0.1*i)
```



```
{  
    //--- set level value of chart object  
    object.LevelValue(i,0.1*i);  
}  
}  
}
```


LevelDescription (Get Method)

Gets a description (text) of the level of a graphical object.

```
string LevelDescription(  
    int level      // level number  
) const
```

Parameters

level

[in] Number of a graphical object level.

Return Value

Description (text) of the specified level of a graphical object that is bound to an instance of the class. Returns NULL if there is no bound object or the object has no specified level.

LevelDescription (Set Method)

Sets the description (text) of the specified graphical object level.

```
bool LevelDescription(  
    int level,      // level number  
    string text     // text  
)
```

Parameters

level

[in] Number of a graphical object level.

text

[in] New value of description (text) of the specified graphical object level.

Return Value

true - success, false - cannot change the description (text).

Example:

```
///--- example for CChartObject::LevelDescription  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///---  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        ///--- get level description of chart object  
        string level_description=object.LevelDescription(i);  
        if(level_description=="")
```



```
{  
    ///--- set level description of chart object  
    object.LevelDescription(i,"Level_"+IntegerToString(i));  
}  
}  
}
```


GetInteger

Provides simplified access to the functions of API MQL5 [ObjectGetInteger\(\)](#) to receive values of integer properties (of bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color types) of a graphical object bound to a class instance. There are two versions of the function call:

Getting a property value without checking the correctness

```
long GetInteger (
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,      // integer property ID
    int modifier=-1                          // modifier
) const
```

Parameters

prop_id

[in] ID of the graphical object double property.

modifier=-1

[in] Modifier (index) of a double property.

Return Value

Value of an integer property - success, 0 - cannot receive an integer property.

Getting a property value verifying the correctness of the operation

```
bool GetInteger (
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,      // integer property ID
    int modifier,                             // modifier
    long& value                               // link to a variable
) const
```

Parameters

prop_id

[in] ID of a graphical object integer property.

modifier

[in] Modifier (index) of an integer property.

value

[out] Link to a variable to place an integer property value.

Return Value

true - success, false - cannot get an integer property.

Example:

```
///--- example for CChartObject::GetInteger
#include <ChartObjects\ChartObject.mqh>
///---
void OnStart ()
```



```
{
    CChartObject object;
    //--- get color of chart object by easy method
    printf("Objects color is %s",ColorToString(object.GetInteger(OBJPROP_COLOR),true));
    //--- get color of chart object by classic method
    long color_value;
    if(!object.GetInteger(OBJPROP_COLOR,0,color_value))
    {
        printf("Get integer property error %d",GetLastError());
        return;
    }
    else
        printf("Objects color is %s",color_value);
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get levels width by easy method
        printf("Level %d width is %d",i,object.GetInteger(OBJPROP_LEVELWIDTH,i));
        //--- get levels width by classic method
        long width_value;
        if(!object.GetInteger(OBJPROP_LEVELWIDTH,i,width_value))
        {
            printf("Get integer property error %d",GetLastError());
            return;
        }
        else
            printf("Level %d width is %d",i,width_value);
    }
}
```


SetInteger

Provides simplified access to the functions of API MQL5 [ObjectSetInteger\(\)](#) to change integer properties (of type bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color types) of a graphical object bound to class instance. There are two versions of the function call:

Setting a property value that does not require a modifier

```
bool SetInteger (
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,    // integer property ID
    long value                                // value
)
```

Parameters

prop_id

[in] ID of a graphical object integer property.

value

[in] New value of a changed integer property.

Setting a property value indicating the modifier

```
bool SetInteger (
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,    // integer property ID
    int modifier,                            // modifier
    long value                                // value
)
```

Parameters

prop_id

[in] ID of a graphical object integer property.

modifier

[in] Modifier (index) of an integer property.

value

[in] New value of an integer property.

Return Value

true - success, false - cannot change the integer property.

Example:

```
///--- example for CChartObject::SetInteger
#include <ChartObjects\ChartObject.mqh>
///---
void OnStart ()
{
    CChartObject object;
    ///--- set new color of chart object
```



```
if(!object.SetInteger(OBJPROP_COLOR,clrRed))
{
    printf("Set integer property error %d",GetLastError());
    return;
}
for(int i=0;i<object.LevelsCount();i++)
{
    //--- set levels width
    if(!object.SetInteger(OBJPROP_LEVELWIDTH,i,i))
    {
        printf("Set integer property error %d",GetLastError());
        return;
    }
}
}
```


GetDouble

Provides simplified access to the functions of API MQL5 [ObjectGetDouble\(\)](#) to receive double values (of float and double types) of a graphical object bound to a class instance. There are two versions of the function call:

Getting a property value without checking the correctness

```
double  GetDouble (
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,      // integer property ID
    int      modifier=-1                      // modifier
) const
```

Parameters

prop_id
[in] ID of the graphical object double property.

modifier=-1
[in] Modifier (index) of a double property.

Return Value

Value of a double property - success, [EMPTY_VALUE](#) - cannot receive a double property.

Getting a property value in verifying the correctness of such treatment

```
bool  GetDouble (
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,      // double property ID
    int      modifier,                      // modifier
    double&   value                        // link to a variable
) const
```

Parameters

prop_id
[in] ID of a graphical object double property.

modifier
[in] Modifier (index) of a double property.

value
[out] Link to a variable to place a double property value.

Return Value

true - success, false - cannot get a double property.

Example:

```
//--- example for CChartObject::GetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart ()
```



```
{
    CChartObject object;
    ///---
    for(int i=0;i<object.LevelsCount();i++)
    {
        ///--- get levels value by easy method
        printf("Level %d value=%f",i,object.GetDouble(OBJPROP_LEVELVALUE,i));
        ///--- get levels value by classic method
        double value;
        if(!object.SetDouble(OBJPROP_LEVELVALUE,i,value))
        {
            printf("Get double property error %d",GetLastError());
            return;
        }
        else
            printf("Level %d value=%f",i,value);
    }
}
```


SetDouble

Provides simplified access to the functions of API MQL5 [ObjectSetDouble\(\)](#) to change double properties (of float and double types) of a graphical object bound to a class instance. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool SetDouble (
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,    // double property ID
    double value                             // value
)
```

Parameters

prop_id

[in] ID of a graphical object double property.

value

[in] New value of a changed double property.

Setting a property value indicating the modifier

```
bool SetDouble (
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,    // double property ID
    int modifier,                          // modifier
    double value                             // value
)
```

Parameters

prop_id

[in] ID of a graphical object double property.

modifier

[in] Modifier (index) of a double property.

value

[in] New value of a changed double property.

Return Value

true - success, false - cannot change the double feature.

Example:

```
///--- example for CChartObject::SetDouble
#include <ChartObjects\ChartObject.mqh>
///---
void OnStart ()
{
    CChartObject object;
    ///---
```



```
for(int i=0;i<object.LevelsCount();i++)
{
    //--- set level value of chart object
    if(!object.SetDouble(OBJPROP_LEVELVALUE,i,0.1*i))
    {
        printf("Set double property error %d",GetLastError());
        return;
    }
}
}
```


GetString

Provides simplified access to the functions of API MQL5 [ObjectGetString\(\)](#) for string property values of a graphical object bound to a class instance. There are two versions of a function call:

Getting a property value without checking the correctness

```
string GetString(  
    ENUM_OBJECT_PROPERTY_STRING prop_id,      // string property ID  
    int modifier=-1                          // modifier  
) const
```

Parameters

prop_id

[in] ID of graphical object string property.

modifier=-1

[in] Modifier (index) of a string property.

Return Value

Value of a string property - success, "" - cannot receive a string property.

Getting a property value verifying the correctness of such treatment

```
bool GetString(  
    ENUM_OBJECT_PROPERTY_STRING prop_id,      // string property ID  
    int modifier,                          // modifier  
    string& value                          // link to a variable  
) const
```

Parameters

prop_id

[in] ID of a graphical object string property.

modifier

[in] Modifier (index) of a string property.

value

[out] Link to a variable to place a string property value.

Return Value

true - successful, false - cannot get a string property.

Example:

```
//--- example for CChartObject::GetString  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{
```



```
CChartObject object;
string      value;
//--- get name of chart object by easy method
printf("Object name is '%s'",object.GetString(OBJPROP_NAME));
//--- get name of chart object by classic method
if(!object.GetString(OBJPROP_NAME,0,value))
{
    printf("Get string property error %d",GetLastError());
    return;
}
else
    printf("Object name is '%s'",value);
for(int i=0;i<object.LevelsCount();i++)
{
    //--- get levels description by easy method
    printf("Level %d description is '%s'",i,object.GetString(OBJPROP_LEVELTEXT,i));
    //--- get levels description by classic method
    if(!object.GetString(OBJPROP_LEVELTEXT,i,value))
    {
        printf("Get string property error %d",GetLastError());
        return;
    }
    else
        printf("Level %d description is '%s'",i,value);
}
}
```


SetString

Provides simplified access to the functions of API MQL5 [ObjectSetString\(\)](#) for changing string properties of a graphical object bound to a class instance. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool SetString(  
    ENUM_OBJECT_PROPERTY_STRING prop_id,    // string property ID  
    string value                            // value  
)
```

Parameters

prop_id

[in] ID of a graphical object string property.

value

[in] New value of a changed string property.

Setting a property value indicating the modifier

```
bool SetString(  
    ENUM_OBJECT_PROPERTY_STRING prop_id,    // string property ID  
    int modifier,                          // modifier  
    string value                            // value  
)
```

Parameters

prop_id

[in] ID of a graphical object string property.

modifier

[in] Modifier (index) of a string property.

value

[in] New value of a changed string property.

Return Value

true - success, false - cannot change a string property.

Example:

```
///--- example for CChartObject::SetString  
#include <ChartObjects\ChartObject.mqh>  
///---  
void OnStart()  
{  
    CChartObject object;  
    ///--- set new name of chart object  
    if(!object.SetString(OBJPROP_NAME, "MyObject"))
```



```
{
    printf("Set string property error %d",GetLastError());
    return;
}
for(int i=0;i<object.LevelsCount();i++)
{
    ///--- set levels description
    if(!object.SetString(OBJPROP_LEVELTEXT,i,"Level_"+IntegerToString(i))
    {
        printf("Set string property error %d",GetLastError());
        return;
    }
}
}
```


Save

Saves parameters of the object in the file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] Handle of the file previously opened using the function FileOpen (...).

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CChartObject::Save  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CChartObject object=new CChartObject;  
    //--- set object parameters  
    //--- . . .  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!object.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
}
```


Load

Loads the parameters of the object from the file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the file previously opened using the function `FileOpen (...)`.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CChartObject::Load  
#include <ChartObjects\ChartObject.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CChartObject object;  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!object.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use object  
    //--- . . .  
}
```


Type

Gets the graphical object type ID.

```
virtual int Type() const
```

Return Value

Object type ID (0x8888 for [CChartObject](#)).

Example:

```
//--- example for CChartObject::Type
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get object type
    int type=object.Type();
}
```


Line Objects

A group of "Lines" graphical objects.

This section contains the technical details of working with a group of classes of "Lines" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectVLine	"Vertical Line" graphical object
CChartObjectHLine	"Horizontal Line" graphical object
CChartObjectTrend	"Trend Line" graphical object
CChartObjectTrendByAngle	"Trend Line by Angle" graphical object
CChartObjectCycles	"Cyclic Lines" graphical object

See also

[Object types](#), [Graphic objects](#)

CChartObjectVLine

CChartObjectVLine is a class for simplified access to "Vertical Line" graphical object properties.

Description

CChartObjectVLine class provides access to "Vertical Line" object properties.

Declaration

```
class CChartObjectVLine : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Vertical Line" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Vertical Line" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int        window,       // chart window  
    datetime   time          // time coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate of the anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_VLINE for [CChartObjectVLine](#)).

CChartObjectHLine

CChartObjectHLine is a class for simplified access to "Horizontal Line" graphical object properties.

Description

CChartObjectHLine class provides access to "Horizontal Line" object properties.

Declaration

```
class CChartObjectHLine : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Horizontal Line" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Horizontal Line" graphical object.

```
bool Create(  
    long    chart_id,    // chart identifier  
    string  name,        // object name  
    long    window,      // chart window  
    double  price        // price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_HLINE for [CChartObjectHLine](#)).

CChartObjectTrend

CChartObjectTrend is a class for simplified access to "Trend Line" graphical object properties.

Description

CChartObjectTrend class provides access to "Trend Line" object properties.

Declaration

```
class CChartObjectTrend : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectTrend

Direct descendants

[CChartObjectChannel](#), [CChartObjectFibo](#), [CChartObjectFiboChannel](#), [CChartObjectFiboExpansion](#),
[CChartObjectGannFan](#), [CChartObjectGannGrid](#), [CChartObjectPitchfork](#), [CChartObjectRegression](#),
[CChartObjectStdDevChannel](#), [CChartObjectTrendByAngle](#)

Class Methods by Groups

Create	
Create	Creates "Trend Line" graphical object
Properties	
RayLeft	Gets/sets "Ray Left" property
RayRight	Gets/sets "Ray Right" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#),
[Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#),

[Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Trend Line" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int       window,        // chart window  
    datetime  time1,         // 1st time coordinate  
    double    price1,        // 1st price coordinate  
    datetime  time2,         // 2nd time coordinate  
    double    price2         // 2nd price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

RayLeft (Get Method)

Gets the value of "Ray Left" property.

```
bool RayLeft () const
```

Return Value

The value of "Ray Left" property assigned to the class instance. If there is no object assigned, it returns false.

RayLeft (Set Method)

Sets new flag value for the "Ray Left" property.

```
bool RayLeft (  
    bool ray      // flag  
)
```

Parameters

ray

[in] New value of the "Ray Left" property.

Return Value

true - successful, false - cannot change flag.

RayRight (Get Method)

Gets the value of "Ray Right" property.

```
bool RayRight() const
```

Return Value

The value of "Ray Right" property, assigned to the class instance. If there is no object assigned, it returns false.

RayRight (Set Method)

Sets new flag value for the "Ray Right" property.

```
bool RayRight (
    bool ray      // flag
)
```

Parameters

ray

[in] New value of the "Ray Right" property.

Return Value

true - successful, false - cannot change flag.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of a file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_TREND for [CChartObjectTrend](#)).

CChartObjectTrendByAngle

CChartObjectTrendByAngle is a class for simplified access to "Trend Line by Angle" graphical object properties.

Description

CChartObjectTrendByAngle class provides access to "Trend Line by Angle" object properties.

Declaration

```
class CChartObjectTrendByAngle : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy



Direct descendants

[CChartObjectGannLine](#)

Class Methods by Groups

Create	
Create	Creates "Trend Line by Angle" graphical object
Properties	
Angle	Gets/sets "Angle" property
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Trend Line by Angle" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    long      window,        // chart window  
    datetime  time1,         // 1st time coordinate  
    double    price1,        // 1st price coordinate  
    datetime  time2,         // 2nd time coordinate  
    double    price2         // 2nd price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double Angle() const
```

Return Value

The value of "Angle" property assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets new value for the "Angle" property.

```
bool Angle(  
    double angle    // angle  
)
```

Parameters

angle

[in] New value of the "Angle" property.

Return Value

true - successful, false - cannot change the property.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_TRENDBYANGLE for [CChartObjectTrendByAngle](#)).

CChartObjectCycles

CChartObjectCycles is a class for simplified access to "Cyclic Lines" graphical object properties.

Description

CChartObjectCycles provides access to "Cyclic Lines" object properties.

Declaration

```
class CChartObjectCycles : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Cycle Lines" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Cyclic Lines" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    long      window,        // chart window  
    datetime  time1,         // 1st time coordinate  
    double    price1,        // 1st price coordinate  
    datetime  time2,         // 2nd time coordinate  
    double    price2         // 2nd price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_CYCLES for [CChartObjectCycles](#)).

Channel Objects

A group of "Channels" graphical objects.

This section contains the technical details of working with a group of classes of "Channels" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectChannel	"Equidistant Channel" graphical object
CChartObjectRegression	"Linear Regression Channel" graphical object
CChartObjectStdDevChannel	"Standard deviations Channel" graphical object
CChartObjectPitchfork	"Andrew's Pitchfork" graphical object

See also

[Object types](#), [Graphic objects](#)

CChartObjectChannel

CChartObjectChannel is a class for simplified access to "Equidistant Channel" graphical object properties.

Description

CChartObjectChannel class provides access to "Equidistant Channel" object properties.

Declaration

```
class CChartObjectChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Equidistant Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Equidistant Channel" graphic object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // time coordinate of the first anchor point  
    double    price1,        // price coordinate of the first anchor point  
    datetime  time2,         // time coordinate of the second anchor point  
    double    price2,        // price coordinate of the second anchor point  
    datetime  time3,         // time coordinate of the third anchor point  
    double    price3         // price coordinate of the third anchor point  
)
```

Parameters

chart_id

[in] Chart ID (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_CHANNEL for [CChartObjectChannel](#)).

CChartObjectRegression

CChartObjectRegression is a class for simplified access to "Linear Regression Channel" graphical object properties.

Description

CChartObjectRegression class provides access to "Linear Regression Channel" object properties.

Declaration

```
class CChartObjectRegression : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Linear Regression Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Linear Regression Channel" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    long      window,        // chart window  
    datetime   time1,        // first time coordinate  
    datetime   time2         // second time coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_REGRESSION for [CChartObjectRegression](#)).

CChartObjectStdDevChannel

CChartObjectStdDevChannel is a class for simplified access to "Standard Deviation Channel" graphical object properties.

Description

CChartObjectStdDevChannel class provides access to "Standard Deviation Channel" object properties.

Declaration

```
class CChartObjectStdDevChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

[CChartObjectTrend](#)

CChartObjectStdDevChannel

Class Methods by Groups

Create	
Create	Creates "Standard Deviation Channel" graphical object
Properties	
Deviations	Gets/sets "Deviation" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Standard Deviation Channel" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int        window,       // chart window  
    datetime   time1,        // first time coordinate  
    datetime   time2,        // second time coordinate  
    double     deviation     // deviation  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

deviation

[in] Numerical value for "Deviation" property.

Return Value

true - successful, false - error.

Deviation (Get Method)

Gets "Deviation" property value.

```
double Deviation() const
```

Return Value

Value of "Deviation" property assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Deviation (Set Method)

Sets "Deviation" property value.

```
bool Deviation(  
    double deviation    // deviation  
)
```

Parameters

deviation

[in] New value for "Deviation" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_STDDEVCHANNEL for [CChartObjectStdDevChannel](#)).

CChartObjectPitchfork

CChartObjectPitchfork is a class for simplified access to "Andrew's Pitchfork" graphical object properties.

Description

CChartObjectPitchfork class provides access to "Andrew's Pitchfork" object properties.

Declaration

```
class CChartObjectPitchfork : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Andrew's Pitchfork" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Andrew's Pitchfork" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    long      window,        // chart window  
    datetime  time1,         // time coordinate of the first anchor point  
    double    price1,        // price coordinate of the first anchor point  
    datetime  time2,         // time coordinate of the second anchor point  
    double    price2,        // price coordinate of the second anchor point  
    datetime  time3,         // time coordinate of the third anchor point  
    double    price3         // price coordinate of the third anchor point  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] Unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_PITCHFORK for [CChartObjectPitchfork](#)).

Gann Tools

A group "Gann Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Gann Tools" graphical objects and the description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectGannLine	"Gann Line" graphical objects
CChartObjectGannFan	"Gann Fan" graphical objects
CChartObjectGannGrid	"Gann Grid" graphical objects

See also

[Object types](#), [Graphic objects](#)

CChartObjectGannLine

CChartObjectGannLine is a class for simplified access to "Gann Line" graphical object properties.

Description

CChartObjectGannLine class provides access to "Gann Line" object properties.

Declaration

```
class CChartObjectGannLine : public CChartObjectTrendByAngle
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy

```

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectTrendByAngle
        CChartObjectGannLine

```

Class Methods by Groups

Create	
Create	Creates "Gann Line" graphical objects
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

Methods inherited from class CChartObjectTrendByAngle

[Angle](#), [Angle](#), [Create](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Gann Line" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    ppb            // pips per bar  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns [EMPTY_VALUE](#).

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool PipsPerBar(  
    double ppb      // pips per bar  
)
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_GANLINE for [CChartObjectGannLine](#)).

CChartObjectGannFan

CChartObjectGannFan is a class for simplified access to "Gann Fan" graphical object properties.

Description

CChartObjectGannFan class provides access to "Gann Fan" object properties.

Declaration

```
class CChartObjectGannFan : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Gann Fan" graphical object
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Downtrend	Gets/sets "Downtrend" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Gann Fan" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    ppb            // pips per bar  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns [EMPTY_VALUE](#).

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool PipsPerBar (
    double ppb      // pips per bar
)
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Downtrend (Get Method)

Gets the value of "Downtrend" flag.

```
bool Downtrend() const
```

Return Value

Value of the "Downtrend" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool Downtrend(  
    bool downtrend    // flag value  
)
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Return Value

true - successful, false - cannot change the flag.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_GANNFAN for [CChartObjectGannFan](#)).

CChartObjectGannGrid

CChartObjectGannGrid is a class for simplified access to "Gann Grid" graphical object properties.

Description

CChartObjectGannGrid class provides access to "Gann Grid" object properties.

Declaration

```
class CChartObjectGannGrid : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

[CChartObjectTrend](#)

CChartObjectGannGrid

Class Methods by Groups

Create	
Create	Creates "Gann Grid" graphical object
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Downtrend	Gets/sets "Downtrend" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Gann Grid" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    ppb            // pips per bar  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns [EMPTY_VALUE](#).

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool PipsPerBar(  
    double ppb      // Pips per bar  
)
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Downtrend (Get Method)

Gets the value of "Downtrend" property.

```
bool Downtrend() const
```

Return Value

Value of "Downtrend" property of the object assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool Downtrend(  
    bool downtrend    // flag value  
)
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphic object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_GANNGRID for [CChartObjectGannGrid](#)).

Fibonacci Tools

A group of "Fibonacci Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Fibonacci Tools" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
<u>CChartObjectFibo</u>	"Fibonacci Retracement" graphical object
<u>CChartObjectFiboTimes</u>	"Fibonacci Time Zones" graphical object
<u>CChartObjectFiboFan</u>	"Fibonacci Fan" graphical object
<u>CChartObjectFiboArc</u>	"Fibonacci Arc" graphical object
<u>CChartObjectFiboChannel</u>	"Fibonacci Channel" graphical object
<u>CChartObjectFiboExpansion</u>	"Fibonacci Expansion" graphical object

See also

[Object types](#), [Graphic objects](#)

CChartObjectFibo

CChartObjectFibo is a class for simplified access to "Fibonacci Retracement" graphical object properties.

Description

CChartObjectFibo class provides access to "Fibonacci Retracement" object properties.

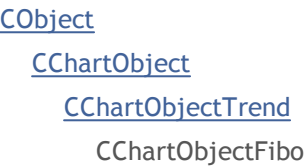
Declaration

```
class CChartObjectFibo : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Fibonacci Retracement" graphical objects
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Retracement" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string     name,       // object name  
    int       window,      // chart window  
    datetime  time1,       // first time coordinate  
    double    price1,      // first price coordinate  
    datetime  time2,       // second time coordinate  
    double    price2       // second price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_FIBO for [CChartObjectFibo](#)).

CChartObjectFiboTimes

CChartObjectFiboTimes is a class for simplified access to "Fibonacci Time Zones" graphical object properties.

Description

CChartObjectFiboTimes class provides access to "Fibonacci Time Zones" object properties.

Declaration

```
class CChartObjectFiboTimes : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Fibonacci Time Zones" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Time Zones" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    price2         // second price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_FIBOTIMES for [CChartObjectFiboTimes](#)).

CChartObjectFiboFan

CChartObjectFiboFan is a class for simplified access to "Fibonacci Fan" graphical object properties.

Description

CChartObjectFiboFan class provides access to "Fibonacci Fan" object properties.

Declaration

```
class CChartObjectFiboFan : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Fibonacci Fan" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Fan" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    price2         // second price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_FIBOFAN for [CChartObjectFiboFan](#)).

CChartObjectFiboArc

CChartObjectFiboArc is a class for simplified access to "Fibonacci Arc" graphical object properties.

Description

CChartObjectFiboArc class provides access to "Fibonacci Arc" object properties.

Declaration

```
class CChartObjectFiboArc : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectFiboArc

Class Methods by Groups

Create	
Create	Creates "Fibonacci Arc" graphical object
Properties	
Scale	Gets/sets "Scale" property
Ellipse	Gets/sets "Ellipse" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Arc" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    price2,        // second price coordinate  
    double    scale          // scale  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

scale

[in] Scale.

Return Value

true - successful, false - error.

Scale (Get Method)

Gets the value of "Scale" property.

```
double Scale() const
```

Return Value

Value of "Scale" property of the object assigned to the class instance. If there is no object assigned, it returns [EMPTY_VALUE](#).

Scale (Set Method)

Sets new value for "Scale" property.

```
bool Scale(  
    double scale    // scale  
)
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

Ellipse (Get Method)

Gets the value of "Ellipse" flag.

```
bool Ellipse() const
```

Return Value

Value of "Ellipse" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

Ellipse (Set Method)

Sets "Ellipse" flag value.

```
bool Ellipse(  
    bool ellipse    // flag value  
)
```

Parameters

ellipse

[in] New value for "Ellipse" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_FIBOARC for [CChartObjectFiboArc](#)).

CChartObjectFiboChannel

CChartObjectFiboChannel is a class for simplified access to "Fibonacci Channel" graphical object properties.

Description

CChartObjectFiboChannel class provides access to "Fibonacci Channel" object properties.

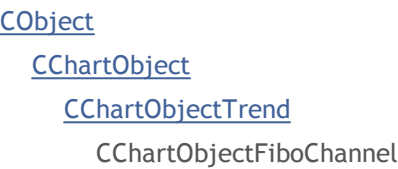
Declaration

```
class CChartObjectFiboChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Fibonacci Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Channel" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int       window,        // chart window  
    datetime   time1,        // first time coordinate  
    double     price1,        // first price coordinate  
    datetime   time2,        // second time coordinate  
    double     price2,        // second price coordinate  
    datetime   time3,        // third time coordinate  
    double     price3         // third price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_FIBOCHANNEL for [CChartObjectFiboChannel](#)).

CChartObjectFiboExpansion

CChartObjectFiboExpansion is a class for simplified access to "Fibonacci Expansion" graphical object properties.

Description

CChartObjectFiboExpansion class provides access to "Fibonacci Expansion" object properties.

Declaration

```
class CChartObjectFiboExpansion : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Fibonacci Expansion" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Fibonacci Expansion" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    int       window,        // chart window  
    datetime   time1,        // first time coordinate  
    double     price1,        // first price coordinate  
    datetime   time2,        // second time coordinate  
    double     price2,        // second price coordinate  
    datetime   time3,        // third time coordinate  
    double     price3         // third price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_EXPANSION for [CChartObjectFiboExpansion](#)).

Elliott Tools

A group of "Elliott Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Elliott Tools" graphical objects.

Class name	Object
<u>CChartObjectElliottWave3</u>	"Correcting Wave" graphical object
<u>CChartObjectElliottWave5</u>	"Impulse Wave" graphical object

See also

[Object types](#), [Graphic objects](#)

CChartObjectElliottWave3

CChartObjectElliottWave3 is a class for simplified access to "Correcting Wave" graphical object properties.

Description

CChartObjectElliottWave3 class provides access to "Correcting Wave" object properties.

Declaration

```
class CChartObjectElliottWave3 : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectElliottWave3

Direct descendants

[CChartObjectElliottWave5](#)

Class Methods by Groups

Create	
Create	Creates "Correcting Wave" graphical object
Properties	
Degree	Gets/sets "Degree" property
Lines	Gets/sets "Lines" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#),

[Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Correcting Wave" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string    name,          // object name  
    int       window,        // chart window  
    datetime  time1,         // first time coordinate  
    double    price1,        // first price coordinate  
    datetime  time2,         // second time coordinate  
    double    price2,        // second price coordinate  
    datetime  time3,         // third time coordinate  
    double    price3         // third price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Time coordinate of the third anchor point.

Return Value

true - successful, false - error.

Degree (Get Method)

Gets the value of "Degree" property.

```
ENUM_ELLIOT_WAVE_DEGREE Degree() const
```

Return Value

Value of "Degree" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Degree (Set Method)

Sets new value for "Degree" property.

```
bool Degree(  
    ENUM_ELLIOT_WAVE_DEGREE degree // property value  
)
```

Parameters

degree

[in] New value for "Degree" property.

Return Value

true - successful, false - cannot change the property.

Lines (Get Method)

Gets the value of "Lines" property.

```
bool Lines() const
```

Return Value

Value of "Lines" property of the object assigned to the class instance. If there is no object assigned, it returns false.

Lines (Set Method)

Sets new value for "Lines" property.

```
bool Lines(  
    bool lines    // flag value  
)
```

Parameters

lines

[in] New value for "Lines" property.

Return Value

true - successful, false - cannot change the flag.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_ELLIOTWAVE3 for [CChartObjectElliottWave3](#)).

CChartObjectElliottWave5

CChartObjectElliottWave5 is a class for simplified access to "Impulse Wave" graphical object properties.

Description

CChartObjectElliottWave5 class provides access to "Impulse Wave" object properties.

Declaration

```
class CChartObjectElliottWave5 : public CChartObjectElliottWave3
```

Title

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Impulse Wave" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectElliottWave3

[Degree](#), [Degree](#), [Lines](#), [Lines](#), [Create](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Impulse Wave" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string     name,       // object name  
    int       window,      // chart window  
    datetime  time1,       // first time coordinate  
    double    price1,      // first price coordinate  
    datetime  time2,       // second time coordinate  
    double    price2,      // second price coordinate  
    datetime  time3,       // third time coordinate  
    double    price3,      // third price coordinate  
    datetime  time4,       // fourth time coordinate  
    double    price4,      // fourth price coordinate  
    datetime  time5,       // fifth time coordinate  
    double    price5,      // fifth price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

time4

[in] Time coordinate of the fourth anchor point.

price4

[in] Price coordinate of the fourth anchor point.

time5

[in] Time coordinate of the fifth anchor point.

price5

[in] Price coordinate of the fifth anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_ELLIOTWAVE5 for [CChartObjectElliottWave5](#)).

Shape Objects

A group of "Shapes" graphical objects.

This section contains the technical details of working with a group of classes of "Shapes" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
<u>CChartObjectRectangle</u>	"Rectangle" graphical objects
<u>CChartObjectTriangle</u>	"Triangle" graphical objects
<u>CChartObjectEllipse</u>	"Ellipse" graphical objects

See also

[Object types](#), [Graphic objects](#)

CChartObjectRectangle

CChartObjectRectangle is a class for simplified access to "Rectangle" graphical object properties.

Description

CChartObjectRectangle class provides access to "Rectangle" object properties.

Declaration

```
class CChartObjectRectangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Rectangle" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Rectangle" graphical object.

```
bool Create(  
    long      chart_id,      // chart identifier  
    string     name,         // object name  
    long      window,        // chart window  
    datetime   time1,        // first time coordinate  
    double     price1,       // first price coordinate  
    datetime   time2,        // second time coordinate  
    double     price2        // second price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_RECTANGLE for [CChartObjectRectangle](#)).

CChartObjectTriangle

CChartObjectTriangle is a class for simplified access to "Triangle" graphical object properties.

Description

CChartObjectTriangle class provides access to "Triangle" object properties.

Declaration

```
class CChartObjectTriangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Triangle" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Triangle" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string     name,       // object name  
    long      window,      // chart window  
    datetime  time1,       // first time coordinate  
    double    price1,      // first price coordinate  
    datetime  time2,       // second time coordinate  
    double    price2,      // second price coordinate  
    datetime  time3,       // third time coordinate  
    double    price3       // third price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_TRIANGLE for [CChartObjectTriangle](#)).

CChartObjectEllipse

CChartObjectEllipse is a class for simplified access to "Ellipse" graphical object properties.

Description

CChartObjectEllipse class provides access to "Ellipse" object properties.

Declaration

```
class CChartObjectEllipse : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Ellipse" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

See also

[Object types](#), [Graphic objects](#)

Create

Creates "Ellipse" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string     name,       // object name  
    int       window,      // chart window  
    datetime  time1,       // first time coordinate  
    double    price1,      // first price coordinate  
    datetime  time2,       // second time coordinate  
    double    price2,      // second price coordinate  
    datetime  time3,       // third time coordinate  
    double    price3       // third price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int Type() const
```

Return Value

Object type identifier (OBJ_ELLIPSE for [CChartObjectEllipse](#)).

Arrow Objects

Group for Arrows graphical objects.

This section contains the technical details of working with a group of classes of "Arrow" graphical objects and a description of the relevant components of the MQL5 Standard Library. In essence, the arrow is a certain icon which is associated with a specific code. There are two types of "Arrow" graphical object to display icons on the charts:

- "Arrow" object, which allows you to specify the code of the icon displayed by the object.
- Group of objects to display a certain type of icons (corresponding to a certain fixed code).

Class for working with arrows displaying arbitrary code icons

Class name	Name of the arrow object
CChartObjectArrow	Arrow

Classes for working arrows displaying a fixed code icon

Class name	Name of the arrow object
CChartObjectArrowCheck	Check
CChartObjectArrowDown	Arrow Up
CChartObjectArrowUp	Arrow Down
CChartObjectArrowStop	Stop Sign
CChartObjectArrowThumbDown	Thumbs Up
CChartObjectArrowThumbUp	Thumbs Down
CChartObjectArrowLeftPrice	Left Price Label
CChartObjectArrowRightPrice	Right Price Label

See also

[Object types](#), [Methods of Object Binding](#), [Graphic objects](#)

CChartObjectArrow

CChartObjectArrow is a class for simplified access to "Arrow" graphical object properties.

Description

CChartObjectArrow class provides access to common properties of "Arrow" objects to all of its descendants.

Declaration

```
class CChartObjectArrow : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectArrow

Direct descendants

CChartObjectArrowCheck, CChartObjectArrowDown, CChartObjectArrowLeftPrice,
CChartObjectArrowRightPrice, CChartObjectArrowStop, CChartObjectArrowThumbDown,
CChartObjectArrowThumbUp, CChartObjectArrowUp

Class Methods by Groups

Create	
Create	Creates "Arrow" graphical object
Properties	
ArrowCode	Gets/sets "Arrow Code" property
Anchor	Gets/sets "Anchor" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Methods of objects binding](#), [Graphic objects](#)

Create

Creates "Arrow" graphical object.

```
bool Create(  
    long      chart_id,      // chart ID  
    string     name,         // object Name  
    int       window,        // chart Window  
    datetime   time,         // time  
    double     price,        // price  
    char       code          // arrow code  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique object name.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate.

price

[in] Price coordinate.

code

[in] "Arrow" code (Wingdings).

Return Value

true - success, false - error.

Example:

```
///--- example for CChartObjectArrow::Create  
#include <ChartObjects\ChartObjectsArrows.mqh>  
///---  
void OnStart()  
{  
    CChartObjectArrow arrow;  
    ///--- set object parameters  
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);  
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))  
    {  
        ///--- arrow create error  
        printf("Arrow create: Error %d!",GetLastError());  
        ///---  
    }  
}
```



```
        return;  
    }  
    //--- use arrow  
    //--- . . .  
}
```


ArrowCode (Get Method)

Gets "Arrow" symbol code.

```
char ArrowCode() const
```

Return Value

"Arrow" symbol code of the object assigned to the class instance. If there is no object assigned, it returns 0.

ArrowCode (Set Method)

Sets symbol code for "Arrow"

```
bool ArrowCode (
    char code      // code value
)
```

Parameters

code

[in] New value for "arrow" code (Wingdings).

Return Value

true - success, false - cannot change the code.

Example:

```
//--- example for CChartObjectArrow::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    CChartObjectArrow arrow;
    char code=181;
    //--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,code))
    {
        //--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
        //---
        return;
    }
    //--- change the code of arrow
    //--- . . .
    //--- get code of arrow
    if(arrow.ArrowCode()!=code)
    {
        //--- set code of arrow
```



```
        arrow.ArrowCode(code);  
    }  
    //--- use arrow  
    //--- . . .  
}
```


Anchor (Get Method)

Gets anchor type of the "Arrow" object

```
ENUM_ARROW_ANCHOR Anchor() const
```

Return Value

Anchor type of the "Arrow" object assigned to the class instance (to the chart). If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets anchor type for the "Arrow" object

```
bool Anchor(
    ENUM_ARROW_ANCHOR anchor // anchor type
)
```

Parameters

anchor

[in] New anchor type value

Return Value

true - successful, false - cannot change the anchor type.

Example:

```
//--- example for CChartObject::Anchor
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    CChartObjectArrow arrow;
    ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM;
    //--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
    {
        //--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
        //---
        return;
    }
    //--- get anchor of arrow
    if(arrow.Anchor()!=anchor)
    {
        //--- set anchor of arrow
        arrow.Anchor(anchor);
    }
```



```
//--- use arrow  
//--- . . .  
}
```


Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Example:

```
//--- example for CChartObjectArrow::Save  
#include <ChartObjects\ChartObjectsArrows.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CChartObjectArrow arrow;  
    //--- set object parameters  
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);  
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))  
    {  
        //--- arrow create error  
        printf("Arrow create: Error %d!",GetLastError());  
        //---  
        return;  
    }  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!arrow.Save(file_handle))  
        {  
            //--- file save error  
            printf("File save: Error %d!",GetLastError());  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
}
```


Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of file previously opened using the FileOpen(...) function.

Return Value

true - success, false - error.

Example:

```
//--- example for CChartObjectArrow::Load  
#include <ChartObjects\ChartObjectsArrows.mqh>  
//---  
void OnStart()  
{  
    int file_handle;  
    CChartObjectArrow arrow;  
    //--- open file  
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);  
    if(file_handle>=0)  
    {  
        if(!arrow.Load(file_handle))  
        {  
            //--- file load error  
            printf("File load: Error %d!",GetLastError());  
            FileClose(file_handle);  
            //---  
            return;  
        }  
        FileClose(file_handle);  
    }  
    //--- use arrow  
    //--- . . .  
}
```


Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (for example, OBJ_ARROW for [CChartObjectArrow](#))

Example:

```
//--- example for CChartObjectArrow::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    CChartObjectArrow arrow;
    //--- get arrow type
    int type=arrow.Type();
}
```


Arrows with fixed code

"Arrows with fixed code" are classes for simplified access to the properties of the following graphical objects:

Class name	Arrow object name
CChartObjectArrowCheck	"Arrow Check"
CChartObjectArrowDown	"Arrow Down"
CChartObjectArrowUp	"Arrow Up"
CChartObjectArrowStop	"Arrow Stop"
CChartObjectArrowThumbDown	"Good" ("Thumbs up")
CChartObjectArrowThumbUp	"Bad" ("Thumbs down")
CChartObjectArrowLeftPrice	"Left price" arrow
CChartObjectArrowRightPrice	"Right price" arrow

Description

"Arrows with fixed code" classes provide access to the object properties.

Declarations

```

class CChartObjectArrowCheck      : public CChartObjectArrow;
class CChartObjectArrowDown      : public CChartObjectArrow;
class CChartObjectArrowUp        : public CChartObjectArrow;
class CChartObjectArrowStop      : public CChartObjectArrow;
class CChartObjectArrowThumbDown : public CChartObjectArrow;
class CChartObjectArrowThumbUp   : public CChartObjectArrow;
class CChartObjectArrowLeftPrice : public CChartObjectArrow;
class CChartObjectArrowRightPrice : public CChartObjectArrow;
```

Title

<ChartObjects\ChartObjectsArrows.mqh>

Class Methods by Groups

Create	
Create	Creates the graphical object matching the class
Properties	
ArrowCode	"Stub" for symbol code change method
Input/output	
virtual Type	Virtual method of identification

See also

[Object types](#), [Methods of object binding](#), [Graphic objects](#)

Create

Creates "Arrow with fixed code" graphical object.

```
bool Create(
    long      chart_id,      // chart ID
    string     name,         // object Name
    int       window,        // chart Window
    datetime   time,         // time
    double     price         // price
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] Unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate.

price

[in] Price coordinate.

Return Value

true - successful, false - error.

Example:

```
//--- example for CChartObjectArrowCheck::Create
//--- example for CChartObjectArrowDown::Create
//--- example for CChartObjectArrowUp::Create
//--- example for CChartObjectArrowStop::Create
//--- example for CChartObjectArrowThumbDown::Create
//--- example for CChartObjectArrowThumbUp::Create
//--- example for CChartObjectArrowLeftPrice::Create
//--- example for CChartObjectArrowRightPrice::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    //--- for example, take CChartObjectArrowCheck
    CChartObjectArrowCheck arrow;
    //--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
```



```
{
    //--- arrow create error
    printf("Arrow create: Error %d!", GetLastError());
    //---
    return;
}
//--- use arrow
//--- . . .
}
```


ArrowCode

Prohibits "Arrow" symbol code changes.

```
bool ArrowCode(  
    char code    // code value  
)
```

Parameters

code

[in] Any value

Return Value

Always false.

Example:

```
//--- example for CChartObjectArrowCheck::ArrowCode  
//--- example for CChartObjectArrowDown::ArrowCode  
//--- example for CChartObjectArrowUp::ArrowCode  
//--- example for CChartObjectArrowStop::ArrowCode  
//--- example for CChartObjectArrowThumbDown::ArrowCode  
//--- example for CChartObjectArrowThumbUp::ArrowCode  
//--- example for CChartObjectArrowLeftPrice::ArrowCode  
//--- example for CChartObjectArrowRightPrice::ArrowCode  
#include <ChartObjects\ChartObjectsArrows.mqh>  
//---  
void OnStart()  
{  
    //--- for example, take CChartObjectArrowCheck  
    CChartObjectArrowCheck arrow;  
    //--- set object parameters  
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);  
    if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))  
    {  
        //--- arrow create error  
        printf("Arrow create: Error %d!",GetLastError());  
        //---  
        return;  
    }  
    //--- set code of arrow  
    if(!arrow.ArrowCode(181))  
    {  
        //--- it is not error  
        printf("Arrow code can not be changed");  
    }  
    //--- use arrow  
    //--- . . .  
}
```


Type

Returns graphical object type identifier

```
virtual int Type() const
```

Return Value

Object type identifier:

CChartObjectArrowCheck - OBJ_ARROW_CHECK,

CChartObjectArrowDown - OBJ_ARROW_DOWN,

CChartObjectArrowUp - OBJ_ARROW_UP,

CChartObjectArrowStop - OBJ_ARROW_STOP,

CChartObjectArrowThumbDown - OBJ_ARROW_THUMB_DOWN,

CChartObjectArrowThumbUp - OBJ_ARROW_THUMB_UP,

CChartObjectArrowLeftPrice - OBJ_ARROW_LEFT_PRICE,

CChartObjectArrowRightPrice - OBJ_ARROW_RIGHT_PRICE.

Example:

```
//--- example for CChartObjectArrowCheck::Type
//--- example for CChartObjectArrowDown::Type
//--- example for CChartObjectArrowUp::Type
//--- example for CChartObjectArrowStop::Type
//--- example for CChartObjectArrowThumbDown::Type
//--- example for CChartObjectArrowThumbUp::Type
//--- example for CChartObjectArrowLeftPrice::Type
//--- example for CChartObjectArrowRightPrice::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
//--- for example, take CChartObjectArrowCheck
    CChartObjectArrowCheck arrow;
//--- get arrow type
    int type=arrow.Type();
}
```


Control Objects

A group of "Object Controls" graphical objects.

This section contains the technical details of working with a group of classes of "Object Controls" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
<u>CChartObjectText</u>	"Text" graphical object
<u>CChartObjectLabel</u>	"Text Label" graphical object
<u>CChartObjectEdit</u>	"Edit" graphical object
<u>CChartObjectButton</u>	"Button" graphical object
<u>CChartObjectSubChart</u>	"Chart" graphical object
<u>CChartObjectBitmap</u>	"Bitmap" graphical object
<u>CChartObjectBmpLabel</u>	"Bitmap Label" graphical object
<u>CChartObjectRectLabel</u>	"Rectangle Label" graphical object

See also

[Object types](#), [Graphic objects](#)

CChartObjectText

CChartObjectText is a class for simplified access to "Text" graphical object properties.

Description

CChartObjectText class provides access to "Text" object properties.

Declaration

```
class CChartObjectText : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectText

Direct descendants

[CChartObjectLabel](#)

Class Methods by Groups

Create	
Create	Creates "Text" graphical object
Properties	
Angle	Gets/sets "Angle" property
Font	Gets/sets "Font" property
FontSize	Gets/sets "FontSize" property
Anchor	Gets/sets "Anchor" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Derived classes:

- [CChartObjectLabel](#)

See also

[Object types](#), [Object properties](#), [Methods of object binding](#), [Graphic objects](#)

Create

Creates "Text" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string    name,        // object name  
    int       window,      // chart window  
    datetime  time,        // time coordinate  
    double    price        // price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double Angle() const
```

Return Value

Value of "Angle" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets a value for "Angle" property.

```
bool Angle(  
    double angle    // property value  
)
```

Parameters

angle

[in] New value for "Angle" property.

Return Value

true - successful, false - cannot change the property.

Font (Get Method)

Gets the value of "Font" property.

```
string Font() const
```

Return Value

Value of "Font" property of the object assigned to the class instance. If there is no object assigned, it returns "".

Font (Set Method)

Sets new value for "Font" property.

```
bool Font(  
    string font    // property value  
)
```

Parameters

font

[in] New value for "Font" property.

Return Value

true - success, false - cannot change the property.

FontSize (Get Method)

Gets the value of "Font size" property.

```
int FontSize() const
```

Return Value

Value of "FontSize" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

FontSize (Set Method)

Sets new value for "Font size" property.

```
bool FontSize(  
    int size    // property value  
)
```

Parameters

size

[in] New value for "Font size" property.

Return Value

true - successful, false - cannot change the property.

Anchor (Get Method)

Gets the value of "Anchor" property.

```
ENUM_ANCHOR_POINT Anchor() const
```

Return Value

Value of "Anchor" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets new value for "Anchor" property.

```
bool Anchor(  
    ENUM_ANCHOR_POINT anchor    // property value  
)
```

Parameters

anchor

[in] New value for "Anchor" property.

Return Value

true - success, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by the [FileOpen](#) function.

Return Value

true - success, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by the [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_TEXT for [CChartObjectText](#)).

CChartObjectLabel

CChartObjectLabel is a class for simplified access to "Label" graphical object properties.

Description

CChartObjectLabel class provides access to "Label" object properties.

Declaration

```
class CChartObjectLabel : public CChartObjectText
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

[CChartObjectText](#)

CChartObjectLabel

Direct descendants

[CChartObjectEdit](#), [CChartObjectRectLabel](#)

Class Methods by Groups

Create	
Create	Creates "Label" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
Corner	Gets/sets "Corner" property
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectText

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

See also

[Object types](#), [Object properties](#), [Chart angle](#), [Methods of Object Binding](#), [Graphic objects](#)

Create

Creates "Label" graphical object.

```
bool Create(  
    long    chart_id,    // chart identifier  
    string  name,        // object name  
    int     window,      // chart window  
    int     X,           // X coordinate  
    int     Y            // Y coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool X_Distance(  
    int x      // property value  
)
```

Parameters

x

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool Y_Distance(  
    int Y    // property value  
)
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

X_Size

Gets the value of "X_Size" property.

```
int X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool Corner(  
    ENUM_BASE_CORNER corner    // property value  
)
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

Time

Prohibits changes of the time coordinate.

```
bool Time(  
    datetime time    // any value  
)
```

Parameters

time

[in] Any value of datetime type.

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool Price(  
    double price    // any value  
)
```

Parameters

price

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_LABEL for [CChartObjectLabel](#)).

CChartObjectEdit

CChartObjectEdit is a class for simplified access to "Edit" graphical object properties.

Description

CChartObjectEdit class provides access to "Edit" object properties.

Declaration

```
class CChartObjectEdit : public CChartObjectLabel
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

[CChartObjectText](#)

[CChartObjectLabel](#)

CChartObjectEdit

Direct descendants

[CChartObjectButton](#)

Class Methods by Groups

Create	
Create	Creates "Edit" graphical object
Properties	
TextAlign	Gets/sets "TextAlign" property
X_Size	Gets "X Size" property
Y_Size	Gets "Y Size" property
BackColor	Gets/sets "Background Color" property
BorderColor	Gets/sets "Border Color" property
ReadOnly	Gets/sets "Read Only" property
Angle	Gets/sets "Angle" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file

virtual [Type](#)

Virtual method of identification

Methods inherited from class CObjectPrev, Prev, Next, Next, [Compare](#)**Methods inherited from class CChartObject**

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectText[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)**Methods inherited from class CChartObjectLabel**[X_Distance](#), [X_Distance](#), [Y_Distance](#), [Y_Distance](#), [X_Size](#), [Y_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)**See also**[Object types](#), [Object properties](#), [Chart angle](#), [Methods of Object Binding](#), [Graphic objects](#)

Create

Creates "Edit" graphical object.

```
bool Create(  
    long    chart_id,    // chart identifier  
    string  name,        // object name  
    int     window,      // chart window  
    int     X,           // X coordinate  
    int     Y,           // Y coordinate  
    int     sizeX,       // X size  
    int     sizeY        // Y size  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Return Value

true - successful, false - error.

TextAlign (Get method)

Gets the value of "TextAlign" property ([text alignment mode](#)).

```
ENUM_ALIGN_MODE TextAlign() const
```

Return Value

Value of "TextAlign" property of the object assigned to the class instance.

TextAlign (Set method)

Sets the value of "TextAlign" property ([text alignment mode](#)).

```
bool TextAlign(  
    ENUM_ALIGN_MODE align    // property value  
)
```

Parameters

align

[in] New value of "TextAlign" property.

Return Value

true - success, false - cannot change the property.

X_Size

Sets the value for "X_Size" property.

```
bool X_Size(  
    int size    // property value  
)
```

Parameters

size

[in] New value for "X_Size" property.

Return Value

true - success, false - cannot change the property.

Y_Size

Sets the value for "Y_Size" property.

```
bool Y_Size(  
    int size    // property value  
)
```

Parameters

size

[in] New value for "Y_Size" property.

Return Value

true - success, false - cannot change the property.

BackColor (Get Method)

Gets the value of "BackColor" property.

```
color BackColor() const
```

Return Value

Value of "BackColor" property of the object assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BackColor (Set Method)

Sets the value for "BackColor" property.

```
bool BackColor(  
    color new_color    // property value  
)
```

Parameters

new_color

[in] New value for "BackColor" property.

Return Value

true - success, false - cannot change the property.

BorderColor (Get Method)

Gets the value of "Border Color" property.

```
color BorderColor() const
```

Return Value

Value of "Border Color" property of the object assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BorderColor (Set Method)

Sets new value for "Border Color" property.

```
bool BorderColor (
    color new_color      // new property value
)
```

Parameters

new_color

[in] New value for "Border Color" property.

Return Value

true - success, false - cannot change the property.

ReadOnly (Get Method)

Gets the value of "Read Only" property.

```
bool ReadOnly() const
```

Return Value

Value of "Read Only" property of the object assigned to the class instance. If there is no object assigned, it returns false.

ReadOnly (Set Method)

Sets the value for "Read Only" property.

```
bool ReadOnly(  
    const bool flag    // new property value  
)
```

Parameters

flag

[in] New value for "Read Only" property (true means text editing is disabled).

Return Value

true - success, false - cannot change the property.

Angle

Prohibits changes of the "Angle" property.

```
bool Angle(  
    double angle    // any value  
)
```

Parameters

angle

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - success, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - success, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_EDIT for [CChartObjectEdit](#)).

CChartObjectButton

CChartObjectButton is a class for simplified access to "Button" graphical object properties.

Description

CChartObjectButton class provides access to "Button" object properties.

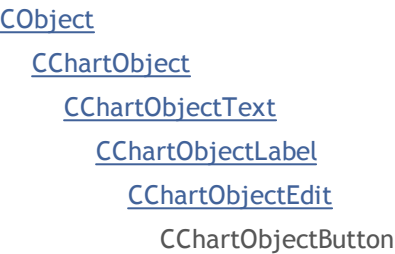
Declaration

```
class CChartObjectButton : public CChartObjectEdit
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy



Direct descendants

CChartObjectPanel

Class Methods by Groups

Create	
Create	Inherited form CChartObjectEdit class
Properties	
State	Gets/sets button state (Pressed/Depressed)
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), Fill, Fill, [Z_Order](#),

[Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectText

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

Methods inherited from class CChartObjectLabel

[X_Distance](#), [X_Distance](#), [Y_Distance](#), [Y_Distance](#), [X_Size](#), [Y_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)

Methods inherited from class CChartObjectEdit

[X_Size](#), [Y_Size](#), [BackColor](#), [BackColor](#), [BorderColor](#), [BorderColor](#), [ReadOnly](#), [ReadOnly](#), [TextAlign](#), [TextAlign](#), [Angle](#), [Create](#)

See also

[Object types](#), [Object properties](#), [Chart angle](#), [Methods of object binding](#), [Graphic objects](#)

State (Get Method)

Gets the value of "State" property.

```
bool State() const
```

Return Value

Value of "State" property of the object assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool State(  
    bool state    // property value  
)
```

Parameters

X

[in] New value for "State" property.

Return Value

true - success, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_BUTTON for [CChartObjectButton](#)).

CChartObjectSubChart

CChartObjectSubChart is a class for simplified access to "Chart" graphical object properties.

Description

CChartObjectSubChart class provides access to "Chart" object properties.

Declaration

```
class CChartObjectSubChart : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectSubChart.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectSubChart

Class Methods by Groups

Create	
Create	Creates "Chart" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
Corner	Gets/sets "Corner" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
Symbol	Gets/sets "Symbol" property
Period	Gets/sets "Period" property
Scale	Gets/sets "Scale" property
DateScale	Gets/sets "Show date scale" property
PriceScale	Gets/sets "Show price scale" property
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file

virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Object properties](#), [Chart angle](#), [Graphic objects](#)

Create

Creates "SubChart" graphical object.

```
bool Create(  
    long    chart_id,    // chart identifier  
    string  name,        // object name  
    int     window,      // chart window  
    int     X,           // X coordinate  
    int     Y,           // Y coordinate  
    int     sizeX,       // X size  
    int     sizeY        // Y size  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool X_Distance(  
    int x    // property value  
)
```

Parameters

x

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool Y_Distance(  
    int Y      // property value  
)
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool Corner(  
    ENUM_BASE_CORNER corner    // property value  
)
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

X_Size (Get Method)

Gets the value of "X_Size" property.

```
int X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Size (Set Method)

Sets new value for "X_Size" property.

```
bool X_Size(  
    int x    // property value  
)
```

Parameters

x

[in] New value for "X_Size" property.

Return Value

true - successful, false - cannot change the property.

Y_Size (Get Method)

Gets the value of "Y_Size" property.

```
int Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size (Set Method)

Sets new value for "Y_Size" property.

```
bool Y_Size(  
    int Y    // property value  
)
```

Parameters

Y

[in] New value for "Y_Size" property.

Return Value

true - successful, false - cannot change the property.

Symbol (Get Method)

Gets the value of "Symbol" property.

```
string Symbol() const
```

Return Value

Value of "Symbol" property of the object assigned to the class instance. If there is no object assigned, it returns "".

Symbol (Set Method)

Sets new value for "Symbol" property.

```
bool Symbol(  
    string symbol    // symbol  
)
```

Parameters

symbol

[in] New value for "Symbol" property.

Return Value

true - successful, false - cannot change the property.

Period (Get Method)

Gets the value of "Period" property.

```
int Period() const
```

Return Value

Value of "Period" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Period (Set Method)

Sets new value for "Period" property.

```
bool Period(  
    int period    // period  
)
```

Parameters

period

[in] New value for "Period" property.

Return Value

true - successful, false - cannot change the property.

Scale (Get Method)

Gets the value of "Scale" property.

```
double Scale() const
```

Return Value

Value of "Scale" property of the object assigned to the class instance. If there is no object assigned, it returns [EMPTY_VALUE](#).

Scale (Set Method)

Sets new value for "Scale" property.

```
bool Scale(  
    double scale    // property value  
)
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

DateScale (Get Method)

Gets the value of "DateScale" flag.

```
bool DateScale() const
```

Return Value

Value of "DateScale" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

DateScale (Set Method)

Sets new value for "DateScale" property.

```
bool DateScale (
    bool scale    // flag value
)
```

Parameters

scale

[in] New value for "DateScale" flag.

Return Value

true - successful, false - cannot change the flag.

PriceScale (Get Method)

Gets the value of "PriceScale" flag.

```
bool PriceScale() const
```

Return Value

Value of "PriceScale" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

PriceScale (Set Method)

Sets new value for "PriceScale" flag.

```
bool PriceScale(  
    bool scale    // flag value  
)
```

Parameters

scale

[in] New value for "PriceScale" flag.

Return Value

true - successful, false - cannot change the flag.

Time

Prohibits changes of the time coordinate.

```
bool Time(  
    datetime time    // any value  
)
```

Parameters

time
[in]

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool Price(  
    double price    // any value  
)
```

Parameters

price
[in]

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_CHART for [CChartObjectSubChart](#)).

CChartObjectBitmap

CChartObjectBitmap is a class for simplified access to "Bitmap" graphical object properties.

Description

CChartObjectBitmap class provides access to "Bitmap" object properties.

Declaration

```
class CChartObjectBitmap : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates "Bitmap" graphical object
Properties	
BmpFile	Gets/sets "BMP Filename" property
X_Offset	Gets/sets "X_Offset" property
Y_Offset	Gets/sets "Y_Offset" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Object properties](#), [Graphic objects](#)

Create

Creates "Bitmap" graphical object.

```
bool Create(  
    long      chart_id,    // chart identifier  
    string     name,       // object name  
    int       window,      // chart window  
    datetime   time,       // time coordinate  
    double     price       // price coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

BmpFile (Get Method)

Gets the value of "BmpFile" property.

```
string BmpFile() const
```

Return Value

Value of "BmpFile" property of the object assigned to the class instance. If there is no object assigned, it returns false.

BmpFile (Set Method)

Sets new value for "BmpFile" property.

```
bool BmpFile(  
    string name    // property value  
)
```

Parameters

name

[in] New value for "BmpFile" property.

Return Value

true - successful, false - cannot change the property.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the upper left corner) of the [CChartObjectBitmap](#) graphical object.

```
int X_Offset() const
```

Return Value

Value of "X_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the upper left corner) of the [CChartObjectBitmap](#) graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool X_Offset(  
    int X        // property value  
)
```

Parameters

X

[in] New value for "X_Offset" property.

Return Value

true - successful, false - cannot change the property.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the upper left corner) of the [CChartObjectBitmap](#) graphical object.

```
int Y_Offset() const
```

Return Value

Value of "Y_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "Y_Offset" property (the upper left corner) of the [CChartObjectBitmap](#) graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool Y_Offset(  
    int Y      // property value  
)
```

Parameters

Y

[in] New value for "Y_Offset" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_BITMAP for [CChartObjectBitmap](#)).

CChartObjectBmpLabel

CChartObjectBmpLabel is a class for simplified access to "Bitmap Label" graphical object properties.

Description

CChartObjectBmpLabel class provides access to "Bitmap Label" object properties.

Declaration

```
class CChartObjectBmpLabel : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

CChartObjectBmpLabel

Class Methods by Groups

Create	
Create	Creates "BmpLabel" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
X_Offset	Gets/sets "X_Offset" property
Y_Offset	Gets/sets "Y_Offset" property
Corner	Gets/sets "Corner" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
BmpFileOn	Gets/sets "BmpFileOn" property for button pressed state (On)
BmpFileOff	Gets/sets "BmpFileOff" property for button depressed state (Off)
State	Gets/sets "Button State" property (Pressed/Depressed)
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change

Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

See also

[Object types](#), [Object properties](#), [Chart angle](#), [Graphic objects](#)

Create

Creates "BmpLabel" graphical object.

```
bool Create(  
    long    chart_id,    // chart identifier  
    string  name,        // object name  
    int     window,      // chart window  
    int     X,           // X coordinate  
    int     Y            // Y coordinate  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool X_Distance(  
    int X      // property value  
)
```

Parameters

X

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool Y_Distance(  
    int Y    // property value  
)
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the upper left corner) of the [CChartObjectBmpLabel](#) graphical object.

```
int X_Offset() const
```

Return Value

Value of "X_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the upper left corner) of the [CChartObjectBitmap](#) graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool X_Offset(  
    int X        // property value  
)
```

Parameters

X

[in] New value for "X_Offset" property.

Return Value

true - successful, false - cannot change the property.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the upper left corner) of the [CCartObjectBmpLabel](#) graphical object.

```
int Y_Offset () const
```

Return Value

Value of "Y_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "Y_Offset" property (the upper left corner) of the [CCartObjectBmpLabel](#) graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool Y_Offset (
    int Y      // property value
)
```

Parameters

Y

[in] New value for "Y_Offset" property.

Return Value

true - successful, false - cannot change the property.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool Corner(  
    ENUM_BASE_CORNER corner    // property value  
)
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

X_Size

Gets the value of "X_Size" property.

```
int X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

BmpFileOn (Get Method)

Gets the value of "BmpFileOn" property.

```
string BmpFileOn() const
```

Return Value

Value of "BmpFileOn" property of the object assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOn (Set Method)

Sets new value for "BmpFileOn" property.

```
bool BmpFileOn (
    string name      // file name
)
```

Parameters

name

[in] New value for "BmpFileOn" property.

Return Value

true - successful, false - cannot change the property.

BmpFileOff (Get Method)

Gets the value of "BmpFileOff" property.

```
string BmpFileOff() const
```

Return Value

Value of "BmpFileOff" property of the object assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOff (Set Method)

Sets new value for "BmpFileOff" property.

```
bool BmpFileOff(  
    string name    // file name  
)
```

Parameters

name

[in] New value for "BmpFileOff" property.

Return Value

true - successful, false - cannot change the property.

State (Get Method)

Gets the value of "State" property.

```
bool State() const
```

Return Value

Value of "State" property of the object assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool State(  
    bool state    // property value  
)
```

Parameters

state

[in] New value for "State" property.

Return Value

true - successful, false - cannot change the property.

Time

Prohibits changes of the time coordinate.

```
bool Time(  
    datetime time    // any value  
)
```

Parameters

time

[in] Any value of datetime type.

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool Price(  
    double price    // any value  
)
```

Parameters

price

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_BITMAP_LABEL for [CChartObjectBmpLabel](#)).

CChartObjectRectLabel

CChartObjectRectLabel is a class for simplified access to "Rectangle Label" graphical object properties.

Description

CChartObjectRectLabel class provides access to "Rectangle Label" object properties.

Declaration

```
class CChartObjectRectLabel : public CChartObjectLabel
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CChartObject](#)

[CChartObjectText](#)

[CChartObjectLabel](#)

CChartObjectRectLabel

Class Methods by Groups

Create	
Create	Creates "RectLabel" graphical object
Properties	
X_Size	Sets the horizontal size
Y_Size	Sets the vertical size
BackColor	Gets/sets the background color
Angle	Prohibits slope angle change
BorderType	Gets/sets type of the border
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z_Order](#), [Z_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Methods inherited from class CChartObjectText

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

Methods inherited from class CChartObjectLabel

[X_Distance](#), [X_Distance](#), [Y_Distance](#), [Y_Distance](#), [X_Size](#), [Y_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)

See also

[Object types](#), [Object properties](#), [Graphic objects](#)

Create

Creates the "CChartObjectRectLabel" graphic object.

```
bool Create(  
    long    chart_id,    // chart ID  
    string  name,        // object name  
    int     window,      // chart window  
    int     X,           // X coordinate  
    int     Y,           // Y coordinate  
    int     sizeX,       // horizontal size  
    int     sizeY        // vertical size  
)
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] Horizontal size.

sizeY

[in] Vertical size.

Return Value

true - successful, false - error.

X_Size

Sets the value of "X_Size" property.

```
bool X_Size(  
    int size    // property value  
)
```

Parameters

size

[in] New horizontal size property value.

Return Value

true - successful, false - cannot change the property.

Note

To get the values of "X_Size" and "Y_Size" properties, use the [X_Size](#) and [Y_Size](#) methods of the parent [CChartObjectLabel](#) class.

Y_Size

Sets the value of "Y_Size" property.

```
bool Y_Size(  
    int size    // property value  
)
```

Parameters

size

[in] New vertical size property value.

Return Value

true - successful, false - cannot change the property.

Note

To get the values of "X_Size" and "Y_Size" properties, use the [X_Size](#) and [Y_Size](#) methods of the parent [CChartObjectLabel](#) class.

BackColor

Gets the background color property value.

```
color BackColor() const
```

Return Value

Background color property value of the object assigned to the class instance. If there is no object assigned, it returns 0.

BackColor

Sets the background color property value.

```
bool BackColor(  
    color new_color    // property value  
)
```

Parameters

new_color

[in] New background color property value.

Return Value

true - successful, false - cannot change the property.

Angle

Prohibits changing the slope angle property.

```
bool Angle(  
    double angle    // any value  
)
```

Parameters

angle

[in] Any value of [double](#) type.

Return Value

Always false.

BorderType

Gets border type property value.

```
int BorderType() const
```

Return Value

Border type property value of the object assigned to the class instance. If there is no object assigned, it returns 0.

BorderType

Sets border type property value.

```
bool BorderType(  
    int type    // property value  
)
```

Parameters

type

[in] New border type property value.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen](#) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int Type() const
```

Return Value

Object type identifier (OBJ_RECTANGLE_LABEL for [CChartObjectRectangleLabel](#)).

CCanvas

CCanvas is a class for simplified creation of custom images.

Description

CCanvas provides creation of a graphical resource (with or without binding to a chart object) and drawing graphic primitives.

Declaration

```
class CCanvas
```

Title

```
#include <Canvas\Canvas.mqh>
```

Inheritance hierarchy

CCanvas

Direct descendants

CChartCanvas, CFlameCanvas

Class methods by groups

Creating	
Attach	Attaches the OBJ_BITMAP_LABEL object to an instance of the CCanvas class
Create	Creates a graphical resource without binding to a chart object
CreateBitmap	Create a graphical resource bound to a chart object
CreateBitmapLabel	Create a graphical resource bound to a chart object
Destroy	Destroys a graphical resource
Properties	
ChartObjectName	Gets the name of a bound chart object
ResourceName	Gets the name of a graphical resource
Width	Gets the width of a graphical resource
Height	Gets the height of a graphical resource
LineStyleSet	Sets the line style
Updates an object on the screen	

Update	Displays changes on the screen
Resize	Resizes a graphical resource
Erasing/Filling with color	
Erase	Erases or fills with the specified color
Data access	
PixelGet	Gets a color of the dot with the specified coordinates
PixelSet	Sets color of the dot with the specified coordinates
Draws primitives	
LineVertical	Draws a vertical line
LineHorizontal	Draws a horizontal line
Line	Draws a freehand line
Polyline	Draws a polyline
Polygon	Draws a polygon
Rectangle	Draws a rectangle
Circle	Draws a circle
Triangle	Draws a triangle
Ellipse	Draws an ellipse
Arc	Draws an ellipse arc
Pie	Draws an ellipse sector
Draws filled primitives	
FillRectangle	Draws a filled rectangle
FillCircle	Draws a filled circle
FillTriangle	Draws a filled triangle
FillPolygon	Draws a filled polygon
FillEllipse	Draws a filled ellipse
Fill	Fills an area
Draws primitives with antialiasing	
PixelSetAA	Draws a pixel
LineAA	Draws a line
PolylineAA	Draws a polyline

<u>PolygonAA</u>	Draws a polygon
<u>TriangleAA</u>	Draws a triangle
<u>CircleAA</u>	Draws a circle
<u>EllipseAA</u>	Draws an ellipse
<u>LineWu</u>	Draws a line
<u>PolylineWu</u>	Draws a polyline
<u>PolygonWu</u>	Draws a polygon
<u>TriangleWu</u>	Draws a triangle
<u>CircleWu</u>	Draws a circle
<u>EllipseWu</u>	Draws an ellipse
<u>LineThick</u>	Draws a segment of a freehand line having a specified width using antialiasing algorithm.
<u>LineThickVertical</u>	Draws a vertical segment of a freehand line having a specified width using antialiasing algorithm.
<u>LineThickHorizontal</u>	Draws a horizontal segment of a freehand line having a specified width using antialiasing algorithm.
<u>PolygonSmooth</u>	Draws a polygon with a specified width using two antialiasing algorithms
<u>PolygonThick</u>	Draws a polygon with a specified width using antialiasing algorithm
<u>PolylineSmooth</u>	Draws a polyline with a specified width using two antialiasing algorithms
<u>PolylineThick</u>	Draws a polyline with a specified width using antialiasing algorithm
Text	
<u>FontSet</u>	Sets font parameters
<u>FontNameSet</u>	Sets font name
<u>FontSizeSet</u>	Sets font size
<u>FontFlagsSet</u>	Sets font flags
<u>FontAngleSet</u>	Sets font slope angle
<u>FontGet</u>	Gets font parameters
<u>FontNameGet</u>	Gets font name
<u>FontSizeGet</u>	Gets font size

<u>FontFlagsGet</u>	Gets font flags
<u>FontAngleGet</u>	Gets font slope angle
<u>TextOut</u>	Displays text
<u>TextWidth</u>	Gets the text width
<u>TextHeight</u>	Gets the text height
<u>TextSize</u>	Gets the text size
Transparency	
<u>TransparentLevelSet</u>	Sets transparency level
Input/output	
<u>LoadFromFile</u>	Reads an image from a BMP file

Attach

Gets the graphical resource from an [OBJ_BITMAP_LABEL](#) object and attaches it to an instance of the CCanvas class.

```
bool Attach(  
    const long      chart_id,           // chart identifier  
    const string    objname,           // object name  
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // color processing method  
    e.)
```

Creates a graphical [resource](#) for an [OBJ_BITMAP_LABEL](#) object and attaches it to an instance of the CCanvas class.

```
bool Attach(  
    const long      chart_id,           // chart identifier  
    const string    objname,           // object name  
    const int       width,             // image width in pixels  
    const int       height,            // image height in pixels  
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // color processing method  
    e.)
```

Parameters

chart_id

[out] Chart identifier.

objname

[in] Name of the graphical object.

width

[in] Image width in the resource.

height

[in] Image height in the resource.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Alpha channel processing method. The alpha channel is ignored by default.

Return Value

true - if successful, false - if failed to attach the object.

Arc

Draws an arc of an ellipse inscribed in a rectangle with corners at (x1,y1) and (x2,y2). The arc boundaries are clipped by lines from the center of the ellipse, which extend to two points with coordinates (x3,y3) and (x4,y4).

```
void Arc(  
    int      x1,      // X coordinate of the upper left corner of the rectangle  
    int      y1,      // Y coordinate of the upper left corner of the rectangle  
    int      x2,      // X coordinate of the bottom right corner of the rectangle  
    int      y2,      // Y coordinate of the bottom right corner of the rectangle  
    int      x3,      // X coordinate of the first point to find the arc boundaries  
    int      y3,      // Y coordinate of the first point to find the arc boundaries  
    int      x4,      // X coordinate of the second point to find the arc boundaries  
    int      y4,      // Y coordinate of the second point to find the arc boundaries  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

x3

[in] X coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y3

[in] Y coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

x4

[in] X coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y4

[in] Y coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

clr

[in] Color in ARGB format. Use the [ColorToARGB\(\)](#) function to convert a color into the ARGB format.

Draws an arc of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry. The arc boundaries are cropped from the ellipse center using rays formed by angles fi3 and fi4.

```
void Arc(
    int      x,          // X coordinate of the ellipse center
    int      y,          // Y coordinate of the ellipse center
    int      rx,         // ellipse radius on the X axis
    int      ry,         // ellipse radius on the Y axis
    int      fi3,        // angle of ray from ellipse center, which defines the first boundary
    int      fi4,        // angle of ray from ellipse center, which defines the second boundary
    const uint clr       // color
);
```

Draws an arc of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry, and also returns the coordinates of the arc boundaries. The arc boundaries are cropped from the ellipse center using rays formed by angles fi3 and fi4.

```
void Arc(
    int      x,          // X coordinate of the ellipse center
    int      y,          // Y coordinate of the ellipse center
    int      rx,         // ellipse radius on the X axis
    int      ry,         // ellipse radius on the Y axis
    int      fi3,        // angle of ray from ellipse center, which defines the first boundary
    int      fi4,        // angle of ray from ellipse center, which defines the second boundary
    int&     x3,         // X coordinate of the first arc boundary
    int&     y3,         // Y coordinate of the first arc boundary
    int&     x4,         // X coordinate of the second arc boundary
    int&     y4,         // Y coordinate of the second arc boundary
    const uint clr       // color
);
```

Parameters

x

[in] X coordinate of the ellipse center.

y

[in] Y coordinate of the ellipse center.

rx

[in] Ellipse radius on the X axis, in pixels.

ry

[in] Ellipse radius on the Y axis, in pixels.

fi3

[in] Angle in radians, which defines the first boundary of the arc.

fi4

[in] Angle in radians, which defines the second boundary of the arc.

x3

[out] Variable to get the X coordinate of the first arc boundary.

y3

[out] Variable to get the Y coordinate of the first arc boundary.

x4

[out] Variable to get the X coordinate of the second arc boundary.

y4

[out] Variable to get the Y coordinate of the second arc boundary.

clr

[in] Color in ARGB format. Use the [ColorToARGB\(\)](#) function to convert a color into the ARGB format.

Examples of calling the class methods:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;

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

void OnStart()
{
    int      Width=600;
    int      Height=400;
    //--- create canvas
    if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
    {
        Print("Error creating canvas: ",GetLastError());
    }
    //--- clear canvas
    canvas.Erase clrWhite);
    //--- draw rectangle
    canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
    //--- draw first arc
    canvas.Arc(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrRed));
    int x1,y1,x2,y2;
    //--- draw second arc
    canvas.Arc(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,x1,y1,x2,y2,ColorToARGB(clrBlue));
    //--- print coordinates of arc
    PrintFormat("First point of arc at (%G,%G), second point of arc at (%G,%G)",x1,y1,x2,y2);
    canvas.CircleAA(x1,y1,3, ColorToARGB(clrRed));
    canvas.CircleAA(x2,y2,3, ColorToARGB(clrBlue));
    //--- show updated canvas
    canvas.Update();
}
```


Pie

Draws a filled sector of an ellipse inscribed in a rectangle with corners at (x1,y1) and (x2,y2). The sector boundaries are clipped by lines from the center of the ellipse, which extend to two points with coordinates (x3,y3) and (x4,y4).

```
void Pie(
    int      x1,        // X coordinate of the upper left corner of the rectangle
    int      y1,        // Y coordinate of the upper left corner of the rectangle
    int      x2,        // X coordinate of the bottom right corner of the rectangle
    int      y2,        // Y coordinate of the bottom right corner of the rectangle
    int      x3,        // X coordinate of the first point to find the arc boundaries
    int      y3,        // Y coordinate of the first point to find the arc boundaries
    int      x4,        // X coordinate of the second point to find the arc boundaries
    int      y4,        // Y coordinate of the second point to find the arc boundaries
    const uint clr,     // line color
    const uint fill_clr // fill color
);
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

x3

[in] X coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y3

[in] Y coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

x4

[in] X coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y4

[in] Y coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

clr

[in] Border color of the sector in the ARGB format.

fill_clr

[in] Fill color of the sector in the ARGB format. Use the [ColorToARGB\(\)](#) function to convert a color into the ARGB format.

Draws a filled sector of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry. The sector boundaries are cropped from the ellipse center by rays formed by angles fi3 and fi4.

```
void Pie(
    int      x,          // X coordinate of the ellipse center
    int      y,          // Y coordinate of the ellipse center
    int      rx,         // ellipse radius on the X axis
    int      ry,         // ellipse radius on the Y axis
    int      fi3,        // angle of ray from ellipse center, which defines the first boundary
    int      fi4,        // angle of ray from ellipse center, which defines the second boundary
    const uint clr,      // line color
    const uint fill_clr // fill color
);
```

Draws a filled sector of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry, and also returns the coordinates of the arc boundaries. The sector boundaries are cropped from the ellipse center by rays formed by angles fi3 and fi4.

```
void Pie(
    int      x,          // X coordinate of the ellipse center
    int      y,          // Y coordinate of the ellipse center
    int      rx,         // ellipse radius on the X axis
    int      ry,         // ellipse radius on the Y axis
    int      fi3,        // angle of ray from ellipse center, which defines the first boundary
    int      fi4,        // angle of ray from ellipse center, which defines the second boundary
    int&     x3,         // X coordinate of the first arc boundary
    int&     y3,         // Y coordinate of the first arc boundary
    int&     x4,         // X coordinate of the second arc boundary
    int&     y4,         // Y coordinate of the second arc boundary
    const uint clr,      // line color
    const uint fill_clr // fill color
);
```

Parameters

x

[in] X coordinate of the ellipse center.

y

[in] Y coordinate of the ellipse center.

rx

[in] Ellipse radius on the X axis, in pixels.

ry

[in] Ellipse radius on the Y axis, in pixels.

fi3

[in] Angle in radians, which defines the first boundary of the arc.

fi4

[in] Angle in radians, which defines the second boundary of the arc.

x3

[out] Variable to get the X coordinate of the first arc boundary.

y3

[out] Variable to get the Y coordinate of the first arc boundary.

x4

[out] Variable to get the X coordinate of the second arc boundary.

y4

[out] Variable to get the Y coordinate of the second arc boundary.

clr

[in] Border color of the sector in the ARGB format.

fill_clr[in] Fill color of the sector in the ARGB format. Use the [ColorToARGB\(\)](#) function to convert a color into the ARGB format.

Examples of calling the class methods:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    int      Width=600;
    int      Height=400;
//--- create canvas
    if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
    {
        Print("Error creating canvas: ",GetLastError());
    }
//--- clear canvas
    canvas.Erase clrWhite);
//--- draw rectangle
    canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- draw first pie
    canvas.Pie(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrBlue),ColorToARGB(c
//--- draw second pie
    canvas.Pie(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,ColorToARGB(clrGreen),Color
//--- show updated canvas
    canvas.Update();
```



```
DebugBreak();  
}
```


FillPolygon

Draws a filled polygon.

```
void FillPolygon(  
    int&      x,      // array with the X coordinates of polygon points  
    int&      y,      // array with the Y coordinates of polygon points  
    const uint clr    // color  
);
```

Parameters

x

[in] Array of the X coordinates of the polygon points.

y

[in] Array of the Y coordinates of the polygon points.

clr

[in] Color in ARGB format.

FillEllipse

Draws a filled ellipse inscribed in a rectangle with the specified coordinates.

```
void FillPolygon(  
    int      x1,      // X coordinate of the upper left corner of the rectangle  
    int      y1,      // Y coordinate of the upper left corner of the rectangle  
    int      x2,      // X coordinate of the bottom right corner of the rectangle  
    int      y2,      // Y coordinate of the bottom right corner of the rectangle  
    const uint clr     // ellipse color  
);
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

clr

[in] Color in ARGB format.

GetDefaultColor

Returns a predefined color by its index.

```
static uint GetDefaultColor(  
    const uint i      // index  
);
```

Parameters

i

[in] Index to get the color.

Return Value

Color.

ChartObjectName

Receives the name of a bound chart object.

```
string ChartObjectName();
```

Return Value

the name of a bound chart object

Circle

Draws a circle

```
void Circle(  
    int      x,      // X coordinate  
    int      y,      // Y coordinate  
    int      r,      // radius  
    const uint clr    // color  
);
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

CircleAA

Draws a circle using antialiasing algorithm

```
void CircleAA(  
    const int    x,        // X coordinate  
    const int    y,        // Y coordinate  
    const double r,        // radius  
    const uint   clr       // color  
) ;
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

Create

Creates a graphical resource without binding to a chart object.

```
virtual bool Create(  
    const string      name,                // name  
    const int         width,              // width  
    const int         height,            // height  
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // format  
);
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See [ResourceCreate\(\)](#) function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

CircleWu

Draws a circle using Wu's anti-aliasing algorithm

```
void CircleWu(  
    const int    x,        // X coordinate  
    const int    y,        // Y coordinate  
    const double r,        // radius  
    const uint   clr       // color  
) ;
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

CreateBitmap

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool CreateBitmap(
    const string      name,           // name
    const datetime    time,           // time
    const double      price,          // price
    const int         width,          // width
    const int         height,         // height
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // format
);
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool CreateBitmap(
    const long        chart_id,       // chart ID
    const int         subwin,         // subwindow number
    const string      name,           // name
    const datetime    time,           // time
    const double      price,          // price
    const int         width,          // width
    const int         height,         // height
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // format
);
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

time

[in] Chart object anchor point time coordinate.

price

[in] Chart object anchor point price coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See [ResourceCreate\(\)](#) function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

CreateBitmapLabel

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool CreateBitmapLabel(
    const string      name,                // name
    const int         x,                  // X coordinate
    const int         y,                  // Y coordinate
    const int         width,              // width
    const int         height,             // height
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // format
);
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool CreateBitmapLabel(
    const long        chart_id,           // chart ID
    const int         subwin,            // subwindow number
    const string      name,              // name
    const int         x,                  // X coordinate
    const int         y,                  // Y coordinate
    const int         width,             // width
    const int         height,            // height
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // format
);
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

x

[in] Chart object anchor point X coordinate.

y

[in] Chart object anchor point Y coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See [ResourceCreate\(\)](#) function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

Destroy

Destroys a graphical resource.

```
void Destroy();
```

Note

If a graphical resource has been bound to a chart object, the latter is deleted.

Ellipse

Draws an ellipse using two points.

```
void Ellipse(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

Erase

Erases or fills with the specified color.

```
void Erase(  
    const uint clr=0    // color  
);
```

Parameters

clr=0

[in] Color in ARGB format.

EllipseAA

Draws an ellipse based on two points using anti-aliasing algorithm.

```
void EllipseAA(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

EllipseWu

Draws an ellipse based on two points using Wu's anti-aliasing algorithm.

```
void EllipseWu(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

Fill

Fills an area.

```
void Fill(  
    int      x,          // X coordinate  
    int      y,          // Y coordinate  
    const uint clr       // color  
);
```

Parameters

x

[in] X coordinate of filling starting point.

y

[in] Y coordinate of filling starting point.

clr

[in] Color in ARGB format.

FillCircle

Draws a filled circle.

```
void FillCircle(  
    int      x,          // X coordinate  
    int      y,          // Y coordinate  
    int      r,          // radius  
    const uint clr       // color  
);
```

Parameters

x

[in] X coordinate of a filled circle center.

y

[in] Y coordinate of a filled circle center.

r

[in] Filled circle radius.

clr

[in] Color in ARGB format.

FillRectangle

Draws a filled rectangle.

```
void FillRectangle(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

FillTriangle

Draws a filled triangle.

```
void FillTriangle(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    int      x3,      // X coordinate  
    int      y3,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

FontAngleGet

Receives font slope angle.

```
uint FontAngleGet ();
```

Return Value

font slope angle

FontAngleSet

Sets font slope angle.

```
bool FontAngleSet(  
    uint angle    // angle  
);
```

Parameters

angle

[in] Font slope angle in tenths of a degree.

Return Value

true - successful, otherwise - false

FontFlagsGet

Receives font flags.

```
uint FontFlagsGet ();
```

Return Value

font flags

FontFlagsSet

Sets font flags.

```
bool FontFlagsSet(  
    uint flags    // flags  
);
```

Parameters

flags

[in] Font creation flags. See [TextSetFont\(\)](#) function description to learn more about the flags.

Return Value

true - successful, otherwise - false

FontGet

Receives the current font parameters.

```
void FontGet(  
    string& name,      // name  
    int& size,         // size  
    uint& flags,       // flags  
    uint& angle        // slope angle  
);
```

Parameters

name

[out] Reference to the variable for returning a font name.

size

[out] Reference to the variable for returning a font size.

flags

[out] Reference to the variable for returning font flags.

angle

[out] Reference to the variable for returning a font slope angle.

FontNameGet

Receives font name.

```
string FontNameGet();
```

Return Value

font name

FontNameSet

Sets font name.

```
bool FontNameSet(  
    string name    // name  
);
```

Parameters

name

[in] Font name. For example, "Arial".

Return Value

true - successful, otherwise - false

FontSet

Sets the current font.

```
bool FontSet(  
    const string name,           // name  
    const int size,             // size  
    const uint flags=0,         // flags  
    const uint angle=0          // angle  
);
```

Parameters

name

[in] Font name. For example, "Arial".

size

[in] Font size. See [TextSetFont\(\)](#) function description to learn more about setting a size.

flags=0

[in] Font creation flags. See [TextSetFont\(\)](#) function description to learn more about the flags.

angle=0

[in] Font slope angle in tenths of a degree.

Return Value

true - successful, otherwise - false

FontSizeGet

Receives font size.

```
int FontSizeGet();
```

Return Value

font size

FontSizeSet

Sets font size.

```
bool FontSizeSet(  
    int size    // size  
);
```

Parameters

size

[in] Font size. See [TextSetFont\(\)](#) function description to learn more about setting a size.

Return Value

true - successful, otherwise - false

Height

Receives the height of a graphical resource.

```
int Height();
```

Return Value

height of a graphical resource

Line

Draws a segment of a freehand line.

```
void Line(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineAA

Draws a segment of a freehand line using antialiasing algorithm.

```
void LineAA(  
    const int   x1,           // X coordinate  
    const int   y1,           // Y coordinate  
    const int   x2,           // X coordinate  
    const int   y2,           // Y coordinate  
    const uint   clr,         // color  
    const uint   style=UINT_MAX // line style  
);
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

LineWu

Draws a segment of a freehand line using Wu's anti-aliasing algorithm.

```
void LineWu(  
    const int   x1,           // X coordinate  
    const int   y1,           // Y coordinate  
    const int   x2,           // X coordinate  
    const int   y2,           // Y coordinate  
    const uint   clr,         // color  
    const uint   style=UINT_MAX // line style  
);
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

LineHorizontal

Draws a segment of a horizontal line.

```
void LineHorizontal(  
    int      x1,      // X coordinate  
    int      x2,      // X coordinate  
    int      y,       // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y

[in] Segment's Y coordinate.

clr

[in] Color in ARGB format.

LineStyleSet

Sets the line style.

```
void LineStyleSet(  
    const uint style    // style  
);
```

Parameters

style

[in] Line style.

Note

The input parameter can have any of ENUM_LINE_STYLE enumeration values. Besides, it is possible to create a custom line drawing style.

LineVertical

Draws a segment of a vertical line.

```
void LineVertical(  
    int      x,          // X coordinate  
    int      y1,         // Y coordinate  
    int      y2,         // Y coordinate  
    const uint clr       // color  
);
```

Parameters

x

[in] Segment's X coordinate.

y1

[in] Y coordinate of the segment's first point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineThickHorizontal

Draws a horizontal segment of a freehand line having a specified width antialiasing.

```
void LineThickHorizontal(  
    const int      x1,           // X coordinate of the segment's first point  
    const int      x2,           // X coordinate of the segment's second point  
    const int      y,           // Y coordinate of the segment  
    const uint     clr,          // color  
    const int      size,         // line width  
    const uint     style,        // line style  
    ENUM_LINE_END  end_style     // line ends style  
)
```

Parameters

x1

[in] X coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y

[in] Segment's Y coordinate.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the `ENUM_LINE_STYLE` enumeration's values or a custom value.

end_style

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values.

LineThickVertical

Draws a vertical segment of a freehand line having a specified width using antialiasing algorithm.

```
void LineThickVertical(  
    const int      x,           // X coordinate of the segment  
    const int      y1,          // Y coordinate of the segment's first point  
    const int      y2,          // Y coordinate of the segment's second point  
    const uint     clr,         // color  
    const int      size,        // line width  
    const uint     style,       // line style  
    ENUM_LINE_END  end_style    // line ends style  
)
```

Parameters

x

[in] Segment's X coordinate.

y1

[in] Y coordinate of the segment's first point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the `ENUM_LINE_STYLE` enumeration's values or a custom value.

end_style

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values.

LineThick

Draws a segment of a freehand line having a specified width using antialiasing algorithm.

```
void LineThick(
    const int      x1,           // X coordinate of the segment's first point
    const int      y1,           // Y coordinate of the segment's first point
    const int      x2,           // X coordinate of the segment's second point
    const int      y2,           // Y coordinate of the segment's second point
    const uint     clr,          // color
    const int      size,         // line width
    const uint     style,        // line style
    ENUM_LINE_END  end_style     // line ends style
)
```

Parameters

- x1*
[in] X coordinate of the segment's first point.
- y1*
[in] Y coordinate of the segment's first point.
- x2*
[in] X coordinate of the segment's second point.
- y2*
[in] Y coordinate of the segment's second point.
- clr*
[in] Color in ARGB format.
- size*
[in] Line width.
- style*
[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.
- end_style*
[in] Line style is one of the ENUM_LINE_END enumeration's values

ENUM_LINE_END

ID	Description
LINE_END_ROUND	Line ends are rounded.
LINE_END_BUTT	Line ends are cut.
LINE_END_SQUARE	A line ends in a filled rectangle.

LoadFromFile

Reads an image from a BMP file.

```
bool LoadFromFile(  
    const string filename    // file name  
);
```

Parameters

filename

[in] File name (including "BMP" extension).

Return Value

true - successful, otherwise - false

PixelGet

Receives color of the point with the specified coordinates.

```
uint PixelGet(  
    const int x,      // X coordinate  
    const int y      // Y coordinate  
);
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

Return Value

Point color in ARGB format.

PixelSet

Sets color of the point with the specified coordinates.

```
void PixelSet(  
    const int   x,           // X coordinate  
    const int   y,           // Y coordinate  
    const uint  clr          // color  
);
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

PixelSetAA

Draws a point using antialiasing algorithm.

```
void PixelSetAA(  
    const double x,      // X coordinate  
    const double y,      // Y coordinate  
    const uint   clr     // color  
);
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

Polygon

Draws a polygon.

```
void Polygon(  
    int&      x[],      // array of X coordinates  
    int&      y[],      // array of Y coordinates  
    const uint clr      // color  
);
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

PolygonAA

Draws a polygon using antialiasing algorithm.

```
void PolygonAA(  
    int&      x[],           // array of X coordinates  
    int&      y[],           // array of Y coordinates  
    const uint clr,          // color  
    const uint style=UINT_MAX // line style  
);
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

PolygonWu

Draws a polygon using Wu's anti-aliasing algorithm.

```
void PolygonWu(  
    int&      x[],           // array of X coordinates  
    int&      y[],           // array of Y coordinates  
    const uint clr,          // color  
    const uint style=UINT_MAX // line style  
);
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

PolygonThick

Draws a polygon with a specified width using antialiasing algorithm.

```
void PolygonThick(  
    const int&    x[],           // array with the X coordinates of polygon points  
    const int&    y[],           // array with the Y coordinates of polygon points  
    const uint    clr,           // color  
    const int     size,          // line width  
    const uint    style,         // line style  
    ENUM_LINE_END end_style      // line ends style  
)
```

Parameters

x[]

[in] Array of X coordinates of polygon points.

y[]

[in] Array of Y coordinates of polygon points.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values.

PolygonSmooth

Draws a polygon with a specified width consecutively using two antialiasing algorithms. First, individual segments are smoothed based on Bezier curves. Then, the raster antialiasing algorithm is applied to the polygon built from these segments to improve the rendering quality.

```
void PolygonSmooth(
    int&          x[],           // array with the X coordinates of p
    int&          y[],           // array with the Y coordinates of p
    const uint    clr,          // color
    const int     size,          // line width
    ENUM_LINE_STYLE style=STYLE_SOLID, // line style
    ENUM_LINE_END end_style=LINE_END_ROUND, // line ends style
    double        tension=0.5,   // antialiasing parameter value
    double        step=10        // length of approximation lines
)
```

Parameters

&x[]

[in] Array of X coordinates of polygon points.

&y[]

[in] Array of Y coordinates of polygon points.

clr

[in] Color in ARGB format.

size

[in] Line width.

style=STYLE_SOLID

[in] Line style is one of the `ENUM_LINE_STYLE` enumeration's values or a custom value.

end_style=LINE_END_ROUND

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values.

tension=0.5

[in] Smoothing parameter value.

step=10

[in] Length of approximating lines.

Polyline

Draws a polyline.

```
void Polyline(  
    int&      x[],      // array of X coordinates  
    int&      y[],      // array of Y coordinates  
    const uint clr      // color  
);
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

PolylineAA

Draws a polyline using antialiasing algorithm.

```
void PolylineAA(  
    int&      x[],           // array of X coordinates  
    int&      y[],           // array of Y coordinates  
    const uint clr,          // color  
    const uint style=UINT_MAX // line style  
);
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

PolylineWu

Draws a polyline using Wu's anti-aliasing algorithm.

```
void PolylineWu(  
    int&      x[],           // array of X coordinates  
    int&      y[],           // array of Y coordinates  
    const uint clr,          // color  
    const uint style=UINT_MAX // line style  
);
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

PolylineSmooth

Draws a polyline with a specified width consecutively using two antialiasing algorithms. First, individual line segments are smoothed based on Bezier curves. Then, the raster antialiasing algorithm is applied to the polyline built from these segments to improve the rendering quality.

```
void PolylineSmooth(  
    const int&      x[],           // array with the X coordinates of po  
    const int&      y[],           // array with the Y coordinates of po  
    const uint      clr,           // color  
    const int       size,          // line width  
    ENUM_LINE_STYLE style=STYLE_SOLID, // line style  
    ENUM_LINE_END   end_style=LINE_END_ROUND, // line ends style  
    double          tension=0.5,    // antialiasing parameter value  
    double          step=10         // approximation step  
)
```

Parameters

&x[]

[in] Array of X coordinates of a polyline.

&y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

size

[in] Line width.

style=STYLE_SOLID

[in] Line style is one of the `ENUM_LINE_STYLE` enumeration's values or a custom value.

end_style=LINE_END_ROUND

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values.

tension=0.5

[in] Smoothing parameter value.

step=10

[in] Approximation step.

PolylineThick

Draws a polyline with a specified width using antialiasing algorithm.

```
void PolylineThick(  
    const int      &x[],           // array with the X coordinates of polyline points  
    const int      &y[],           // array with the Y coordinates of polyline points  
    const uint     clr,           // color  
    const int      size,          // line width  
    const uint     style,         // line style  
    ENUM_LINE_END  end_style      // line ends style  
)
```

Parameters

&x[]

[in] Array of X coordinates of a polyline.

&y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the [ENUM_LINE_END](#) enumeration's values

Rectangle

Draws a rectangle using two points.

```
void Rectangle(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

Resize

Resizes a graphical resource.

```
bool  Resize(  
    const int  width,      // width  
    const int  height     // height  
);
```

Parameters

width

[in] New width of a graphical resource.

height

[in] New height of a graphical resource.

Return Value

true - successful, otherwise - false

Note

When resizing, the previous image is not saved.

ResourceName

Receives the name of a graphical resource.

```
string ResourceName();
```

Return Value

name of a graphical resource

TextHeight

Receives the text height.

```
int TextHeight(  
    const string text    // text  
);
```

Parameters

text

[in] Text for measuring.

Return Value

text height in pixels

Note

The current font is used for measuring the text.

TextOut

Displays text.

```
void TextOut (
    int      x,           // X coordinate
    int      y,           // Y coordinate
    string    text,        // text
    const uint clr,        // color
    uint      alignment=0  // alignment
);
```

Parameters

x

[in] Text anchor's X coordinate.

y

[in] Text anchor's Y coordinate.

text

[in] Text to be displayed.

clr

[in] Color in ARGB format.

alignment=0

[in] Text anchoring method. See [TextOut\(\)](#) function description to learn more about anchoring methods.

Note

The current font is used to display the text.

TextSize

Receives the text size.

```
void TextSize(  
    const string text,      // text  
    int& width,             // width  
    int& height            // height  
);
```

Parameters

text

[in] Text for measuring.

width

[out] Reference to the variable for returning a text width.

height

[out] Reference to the variable for returning a text height.

Note

The current font is used to measure the text.

TextWidth

Receives the text width.

```
int TextWidth(  
    const string text    // text  
);
```

Parameters

text

[in] Text for measuring.

Return Value

text height in pixels

Note

The current font is used to measure the text.

TransparentLevelSet

Sets transparency level.

```
void TransparentLevelSet(  
    const uchar value    // value  
);
```

Parameters

value

[in] New value of the transparency level.

Note

0 stands for full transparency, while 255 - for full opacity.

Setting a transparency level affects all that was previously drawn. The specified transparency level does not affect further constructions.

Triangle

Draws a triangle.

```
void Triangle(  
    int      x1,      // X coordinate  
    int      y1,      // Y coordinate  
    int      x2,      // X coordinate  
    int      y2,      // Y coordinate  
    int      x3,      // X coordinate  
    int      y3,      // Y coordinate  
    const uint clr     // color  
);
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

TriangleAA

Draws a triangle using antialiasing algorithm.

```
void TriangleAA(  
    const int   x1,           // X coordinate  
    const int   y1,           // Y coordinate  
    const int   x2,           // X coordinate  
    const int   y2,           // Y coordinate  
    const int   x3,           // X coordinate  
    const int   y3,           // Y coordinate  
    const uint   clr,         // color  
    const uint   style=UINT_MAX // line style  
);
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

TriangleWu

Draws a triangle using Wu's anti-aliasing algorithm.

```
void TriangleWu(  
    const int   x1,           // X coordinate  
    const int   y1,           // Y coordinate  
    const int   x2,           // X coordinate  
    const int   y2,           // Y coordinate  
    const int   x3,           // X coordinate  
    const int   y3,           // Y coordinate  
    const uint   clr,         // color  
    const uint   style=UINT_MAX // line style  
);
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of [ENUM_LINE_STYLE](#) enumeration's values or a custom value.

Update

Displays changes on the screen.

```
void Update(  
    const bool  redraw=true    // flag  
);
```

Parameters

redraw=true

Flag of a chart redrawing necessity.

Width

Receives the width of a graphical resource.

```
int Width();
```

Return Value

graphical resource width

CChart

CChart is a class for simplified access to "Chart" graphic object properties.

Description

CChart class provides access to "Chart" object properties.

Declaration

```
class CChart : public CObject
```

Title

```
#include <Charts\Chart.mqh>
```

Inheritance hierarchy

CObject

CChart

Class Methods by Groups

Access to protected data	
<u>ChartID</u>	Gets identifier of the chart
General properties	
<u>Mode</u>	Gets/sets the value of "Mode" property (bars, candles, or line)
<u>Foreground</u>	Gets/sets the value of "Foreground" property
<u>Shift</u>	Gets/sets the value of "Shift" property
<u>ShiftSize</u>	Gets/sets the value of "ShiftSize" property (in percents)
<u>AutoScroll</u>	Gets/sets the value of "AutoScroll" property
<u>Scale</u>	Gets/sets the value of "Scale" property
<u>ScaleFix</u>	Gets/sets the value of "ScaleFix" property (fixed chart scale or not)
<u>ScaleFix_11</u>	Gets/sets the value of "ScaleFix_11" property (chart scale is 1:1, or not)
<u>FixedMax</u>	Gets/sets the value of "FixedMax" property (fixed maximal price)
<u>FixedMin</u>	Gets/sets the value of "FixedMin" property (fixed minimal price)

<u>ScalePPB</u>	Gets/sets the value of "ScalePPB" property (scale is "point per bar" or not)
<u>PointsPerBar</u>	Gets/sets the value of "PointsPerBar" property (in points per bar)
Show properties	
<u>ShowOHLC</u>	Gets/sets the value of "ShowOHLC" property
<u>ShowLineBid</u>	Gets/sets the value of "ShowLineBid" property
<u>ShowLineAsk</u>	Gets/sets the value of "ShowLineAsk" property
<u>ShowLastLine</u>	Gets/sets the value of "ShowLastLine" property
<u>ShowPeriodSep</u>	Gets/sets the value of "ShowPeriodSep" property (show period separators)
<u>ShowGrid</u>	Gets/sets the value of "ShowGrid" property
<u>ShowVolumes</u>	Gets/sets the value of "ShowVolumes" property (color for volumes and levels of opened positions)
<u>ShowObjectDescr</u>	Gets/sets the value of "ShowObjectDescr" property (show description for graphic objects)
<u>ShowDateScale</u>	Sets the value of "ShowDateScale" property (date scale of the chart)
<u>ShowPriceScale</u>	Sets the value of "ShowPriceScale" property (price scale of the chart)
Color properties	
<u>ColorBackground</u>	Gets/sets the value of "ColorBackground" property (background color of the chart)
<u>ColorForeground</u>	Gets/sets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart)
<u>ColorGrid</u>	Gets/sets the value of "ColorGrid" property (color of the grid)
<u>ColorBarUp</u>	Gets/sets the value of "ColorBarUp" property (color for bull bars, their shadow and candle body outlines)
<u>ColorBarDown</u>	Gets/sets the value of "ColorBarDown" property (color for bear bars, their shadow and candle body outlines)
<u>ColorCandleBull</u>	Gets/sets the value of "ColorCandleBull" property (body color of the bull candle)

<u>ColorCandleBear</u>	Gets/sets the value of "ColorCandleBear" property (body color of the bear candle)
<u>ColorChartLine</u>	Gets/sets the value of "ColorChartLine" property (color for line chart and Doji candles)
<u>ColorVolumes</u>	Gets/sets the value of "ColorVolumes" property (color for volumes and levels of opened positions)
<u>ColorLineBid</u>	Gets/sets the value of "ColorLineBid" property (color of Bid line)
<u>ColorLineAsk</u>	Gets/sets the value of "ColorLineAsk" property (color of Ask line)
<u>ColorLineLast</u>	Gets/sets the value of "ColorLineLast" property (color of the last deal price line)
<u>ColorStopLevels</u>	Gets/sets the value of "ColorStopLevels" property (color of the SL and TP levels)
Read only properties	
<u>VisibleBars</u>	Gets total number of visible chart bars
<u>WindowsTotal</u>	Gets total number of chart windows, including the chart indicator subwindows
<u>WindowsVisible</u>	Gets visibility flag of the specified chart subwindow
<u>WindowHandle</u>	Gets window handle of the chart (HWND)
<u>FirstVisibleBar</u>	Gets the number of the first visible bar of the chart
<u>WidthInBars</u>	Gets window width in bars.
<u>WidthInPixels</u>	Gets subwindow width in pixels.
<u>HeightInPixels</u>	Gets subwindow height in pixels.
<u>PriceMin</u>	Gets minimal price of the specified subwindow
<u>PriceMax</u>	Gets maximal price of the specified subwindow
Properties	
<u>Attach</u>	Assigns the current chart to the class instance
<u>FirstChart</u>	Assigns the first chart of the client terminal to the class instance
<u>NextChart</u>	Assigns the next chart of the client terminal to the class instance
<u>Open</u>	Opens chart with specified parameters and assign it to the class instance

Detach	Detaches chart from the class instance
Close	Closes chart assigned to the class instance
BringToTop	Show chart on top of other charts
EventObjectCreate	Sets a flag to send notifications of an event of new object creation on a chart
EventObjectDelete	Sets a flag to send notifications of an event of object deletion on a chart
Indicators	
IndicatorAdd	Adds an indicator with the specified handle into a specified chart subwindow
IndicatorDelete	Removes an indicator with a specified name from the specified chart subwindow
IndicatorsTotal	Returns the number of all indicators applied to the specified chart subwindow
IndicatorName	Returns the short name of the indicator on the specified chart subwindow
Navigation	
Navigate	Navigates the chart
Access to MQL5 API	
Symbol	Gets symbol of the chart
Period	Gets period of the chart
Redraw	Redraws chart, assigned to the class instance
GetInteger	The function returns the value of the corresponding object property
SetInteger	Sets new value for the property of the integer type
GetDouble	The function returns the value of the corresponding object property
SetDouble	Sets new value for the property of the double type
GetString	The function returns the value of the corresponding object property
SetString	Sets new value for the property of the string type
SetSymbolPeriod	Changes symbol and period of the chart assigned to the class instance

<u>ApplyTemplate</u>	Applies specified template to the chart
<u>ScreenShot</u>	Creates screenshot of the specified chart and saves it to .gif file
<u>WindowOnDropped</u>	Gets chart subwindow number corresponding to the object (expert or script) drop point
<u>PriceOnDropped</u>	Gets price coordinate corresponding to the object (expert or script) drop point
<u>TimeOnDropped</u>	Gets time coordinate corresponding to the object (expert or script) drop point
<u>XOnDropped</u>	Gets X coordinate corresponding to the object (expert or script) drop point
<u>YOnDropped</u>	Gets Y coordinate corresponding to the object (expert or script) drop point
Input/Output	
virtual <u>Save</u>	Saves object parameters to file
virtual <u>Load</u>	Loads object parameters from file
virtual <u>Type</u>	Gets graphic object type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

ChartID

Returns identifier of the chart.

```
long ChartID() const
```

Return Value

Chart identifier assigned to the chart class instance. If there is no chart assigned, it returns -1.

Mode (Get Method)

Gets the value of "Mode" property (bars, candles, or line).

```
ENUM_CHART_MODE Mode() const
```

Return Value

Value of "Mode" property of the object assigned to the class instance. If there is no chart assigned, it returns [WRONG_VALUE](#).

Mode (Set Method)

Sets new value for "Mode" property (bars, candles, or line).

```
bool Mode(  
    ENUM_CHART_MODE mode    // chart mode  
)
```

Parameters

mode

[in] Chart mode (candles, bars or line) of [ENUM_CHART_MODE](#) enumeration.

Return Value

true - successful, false - cannot change the mode.

Foreground (Get Method)

Gets the value of "Foreground" property.

```
bool Foreground() const
```

Return Value

Value of "Foreground" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

Foreground (Set Method)

Sets new value for "Foreground" property.

```
bool Foreground(  
    bool foreground    // flag value  
)
```

Parameters

foreground

[in] New value for "Foreground" property.

Return Value

true - successful, false - cannot change the property.

Shift (Get Method)

Gets the value of "Shift" property.

```
bool Shift() const
```

Return Value

Value of "Shift" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

Shift (Set Method)

Sets new value for "Shift" property.

```
bool Shift(  
    bool shift    // flag value  
)
```

Parameters

shift

[in] New value for "Shift" property.

Return Value

true - successful, false - cannot change the property.

ShiftSize (Get Method)

Gets the value of "ShiftSize" property (in percents).

```
double ShiftSize() const
```

Return Value

Value of "ShiftSize" property of the chart assigned to the class instance. If there is no chart assigned, it returns [EMPTY_VALUE](#).

ShiftSize (Set Method)

Sets new value for "Shift" property (in percents).

```
bool ShiftSize(  
    double shift_size    // property value  
)
```

Parameters

shift_size

[in] New value for "ShiftSize" property (in percents).

Return Value

true - successful, false - cannot change the property.

AutoScroll (Get Method)

Gets the value of "AutoScroll" property.

```
bool AutoScroll() const
```

Return Value

Value of "AutoScroll" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

AutoScroll (Set Method)

Sets new value for "AutoScroll" property.

```
bool AutoScroll(  
    bool autoscroll    // flag value  
)
```

Parameters

autoscroll

[in] New value for "Autoscroll" property.

Return Value

true - successful, false - cannot change the property.

Scale (Get Method)

Gets the value of "Scale" property.

```
int Scale() const
```

Return Value

Value of "Scale" property of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool Scale(  
    int scale    // property value  
)
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

ScaleFix (Get Method)

Gets the value of "ScaleFix" property (fixed chart scale or not).

```
bool ScaleFix() const
```

Return Value

Value of "ScaleFix" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScaleFix (Set Method)

Sets new value for "ScaleFix" property.

```
bool ScaleFix(  
    bool scale_fix    // property value  
)
```

Parameters

scale_fix

[in] New value for "ScaleFix" property.

Return Value

true - successful, false - cannot change the property.

ScaleFix_11 (Get Method)

Gets the value of "ScaleFix_11" property (chart scale is 1:1, or not).

```
bool ScaleFix_11() const
```

Return Value

Value of "ScaleFix_11" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScaleFix_11 (Set Method)

Sets new value for "ScaleFix_11" property.

```
bool ScaleFix_11(  
    string scale_11    // property value  
)
```

Parameters

scale_11

[in] New value for "ScaleFix_11" property.

Return Value

true - successful, false - cannot change the property.

FixedMax (Get Method)

Gets the value of "FixedMax" property (fixed maximal price).

```
double FixedMax() const
```

Return Value

Value of "FixedMax" property of the chart assigned to the class instance. If there is no chart assigned, it returns [EMPTY_VALUE](#).

FixedMax (Set Method)

Sets the new value for "FixedMax" property.

```
bool FixedMax (
    double max      // fixed maximum
)
```

Parameters

max

[in] New value for "FixedMax" property.

Return Value

true - successful, false - cannot change the property.

FixedMin (Get Method)

Gets the value of "FixedMin" property (fixed minimal price).

```
double FixedMin() const
```

Return Value

Value of "FixedMin" property of the chart assigned to the class instance. If there is no chart assigned, it returns [EMPTY_VALUE](#).

FixedMin (Set Method)

Sets new value for "FixedMin" property.

```
bool FixedMax (
    double min      // fixed minimum
)
```

Parameters

max

[in] New value for "FixedMin" property.

Return Value

true - successful, false - cannot change the property.

PointsPerBar (Get Method)

Gets the value of "PointsPerBar" property (in points per bar).

```
double PointsPerBar() const
```

Return Value

Value of "PointsPerBar" property of the chart assigned to the class instance. If there is no chart assigned, it returns [EMPTY_VALUE](#).

PointsPerBar (Set Method)

Sets new value for "PointsPerBar" property.

```
bool PointsPerBar (
    double ppb      // scale
)
```

Parameters

ppb

[in] New scale (in points per bar).

Return Value

true - successful, false - cannot change the scale.

ScalePPB (Get Method)

Gets the value of "ScalePPB" property (scale is "point per bar" or not).

```
bool ScalePPB() const
```

Return Value

Value of "ScalePPB" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScalePPB (Set Method)

Sets new value for "ScalePPB" property.

```
bool ScalePPB(  
    bool scale_ppb    // flag value  
)
```

Parameters

scale_ppb

[in] New value for "ScalePPB" property.

Return Value

true - successful, false - cannot change the property.

ShowOHLC (Get Method)

Gets the value of "ShowOHLC" property.

```
bool ShowOHLC() const
```

Return Value

Value of "ShowOHLC" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowOHLC (Set Method)

Sets new value for "ShowOHLC" property.

```
bool ShowOHLC(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowOHLC" property.

Return Value

true - success, false - cannot change the property.

ShowLineBid (Get Method)

Gets the value of "ShowLineBid" property.

```
bool ShowLineBid() const
```

Return Value

Value of "ShowLineBid" property of the char, assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineBid (Set Method)

Sets new value for "ShowLineBid" property.

```
bool ShowLineBid(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowLineBid" property.

Return Value

true - successful, false - cannot change the property.

ShowLineAsk (Get Method)

Gets the value of "ShowLineAsk" property.

```
bool ShowLineAsk() const
```

Return Value

Value of "ShowLineAsk" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineAsk (Set Method)

Sets new value for "ShowLineAsk" property.

```
bool ShowLineAsk(  
    bool show // property value  
)
```

Parameters

show

[in] New value for "ShowLineAsk" property.

Return Value

true - successful, false - cannot change the property.

ShowLastLine (Get Method)

Gets the value of "ShowLastLine" property.

```
bool ShowLastLine() const
```

Return Value

Value of "ShowLastLine" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowLastLine (Set Method)

Sets new value for "ShowLastLine" property.

```
bool ShowLastLine (  
    bool show // property value  
)
```

Parameters

show

[in] New value for "ShowLastLine" property.

Return Value

true - successful, false - cannot change the property.

ShowPeriodSep (Get Method)

Gets the value of "ShowPeriodSep" property (show period separators).

```
bool ShowPeriodSep() const
```

Return Value

Value of "ShowPeriodSep" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowPeriodSep (Set Method)

Sets new value for "ShowPeriodSep" property.

```
bool ShowPeriodSep(  
    bool show // property value  
)
```

Parameters

show

[in] New value for "ShowPeriodSep" property.

Return Value

true - successful, false - cannot change the property.

ShowGrid (Get Method)

Gets the value of "ShowGrid" property.

```
bool ShowGrid() const
```

Return Value

Value of "ShowGrid" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowGrid (Set Method)

Sets new value for "ShowGrid" property.

```
bool ShowGrid(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowGrid" property.

Return Value

true - successful, false - cannot change the property.

ShowVolumes (Get Method)

Gets the value of "ShowVolumes" property.

```
bool ShowVolumes() const
```

Return Value

Value of "ShowVolumes" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowVolumes (Set Method)

Sets new value for "ShowVolumes" property.

```
bool ShowVolumes (  
    bool show // property value  
)
```

Parameters

show

[in] New value for "ShowVolumes" property.

Return Value

true - successful, false - cannot change the property.

ShowObjectDescr (Get Method)

Gets the value of "ShowObjectDescr" property (show description for graphic objects).

```
bool ShowObjectDescr() const
```

Return Value

Value of "ShowObjectDescr" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowObjectDescr (Set Method)

Sets new value for "ShowObjectDescr" property.

```
bool ShowObjectDescr(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowObjectDescr" property.

Return Value

true - successful, false - cannot change the property.

ShowDateScale

Sets new value for "ShowDateScale" property.

```
bool ShowDateScale(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowDateScale" property.

Return Value

true - successful, false - cannot change the property.

ShowPriceScale

Sets new value for "ShowPriceScale" property.

```
bool ShowPriceScale(  
    bool show    // property value  
)
```

Parameters

show

[in] New value for "ShowPriceScale" property.

Return Value

true - successful, false - cannot change the property.

ColorBackground (Get Method)

Gets the value of "ColorBackground" property (background color of the chart).

```
color ColorBackground() const
```

Return Value

Value of "ColorBackground" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorBackground (Set Method)

Sets new value for "ColorBackground" property.

```
bool ColorBackground(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New background color.

Return Value

true - successful, false - cannot change the color.

ColorForeground (Get Method)

Gets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart).

```
color ColorForeground() const
```

Return Value

Value of "ColorForeground" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorForeground (Set Method)

Sets new value for "ColorForeground" property (for axes, scale, and OHLC string).

```
bool ColorForeground(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color for axes, scale and OHLC string.

Return Value

true - successful, false - cannot change the color.

ColorGrid (Get Method)

Gets the value of "ColorGrid" property (color of the grid).

```
color ColorGrid() const
```

Return Value

Value of "ColorGrid" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorGrid (Set Method)

Sets new value for "ColorGrid" property.

```
bool ColorGrid(  
    color new_color    // color  
)
```

Parameters

new_color
[in] New grid color.

Return Value

true - successful, false - cannot change the color.

ColorBarUp (Get Method)

Gets the value of "ColorBarUp" property (color for bullish bars, their shadow, and candle body outlines).

```
color ColorBarUp() const
```

Return Value

Value of "ColorBarUp" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorBarUp (Set Method)

Sets new value for "ColorBarUp" property.

```
bool ColorBarUp(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color for bullish bars, their shadow and candle body outlines.

Return Value

true - successful, false - cannot change the color.

ColorBarDown (Get Method)

Gets the value of "ColorBarDown" property (color for bearish bars, their shadow, and candle body outlines).

```
color ColorBarDown() const
```

Return Value

Value of "ColorBarDown" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorBarDown (Set Method)

Sets new value for "ColorBarDown" property.

```
bool ColorBarDown(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color for bearish bars, their shadow, and candle body outlines.

Return Value

true - successful, false - cannot change the color.

ColorCandleBull (Get Method)

Gets the value of "ColorCandleBull" property (body color of the bullish candle).

```
color ColorCandleBull() const
```

Return Value

Value of "ColorCandleBull" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorCandleBull (Set Method)

Sets new value for "ColorCandleBull" property.

```
bool ColorCandleBull(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color of the bullish candle body.

Return Value

true - successful, false - cannot change the color.

ColorCandleBear (Get Method)

Gets the value of "ColorCandleBear" property (body color of the bearish candle).

```
color ColorCandleBear() const
```

Return Value

Value of "ColorCandleBear" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorCandleBear (Set Method)

Sets new value for "ColorCandleBear" property.

```
bool ColorCandleBear(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color of the bearish candle body.

Return Value

true - successful, false - cannot change the color.

ColorChartLine (Get Method)

Gets the value of "ColorChartLine" property (color for line chart and Doji candles).

```
color ColorChartLine() const
```

Return Value

Value of "ColorChartLine" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorChartLine (Set Method)

Sets new value for "ColorChartLine" property.

```
bool ColorChartLine(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color of the chart lines and Doji candles.

Return Value

true - successful, false - cannot change the color.

ColorVolumes (Get Method)

Gets the value of "ColorVolumes" property (color for volumes and levels of opened positions).

```
color ColorVolumes() const
```

Return Value

Value of "ColorVolumes" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorVolumes (Set Method)

Sets new value for "ColorVolumes" property.

```
bool ColorVolumes (
    color new_color    // color
)
```

Parameters

new_color

[in] New color of the volumes and open position levels.

Return Value

true - successful, false - cannot change the color.

ColorLineBid (Get Method)

Gets the value of "ColorLineBid" property (color of Bid line).

```
color ColorLineBid() const
```

Return Value

Value of "ColorLineBid" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorLineBid (Set Method)

Sets new value for "ColorLineBid" property.

```
bool ColorLineBid(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color for Bid line.

Return Value

true - successful, false - cannot change the color.

ColorLineAsk (Get Method)

Gets the value of "ColorLineAsk" property (color of Ask line).

```
color ColorLineAsk() const
```

Return Value

Value of "ColorLineAsk" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorLineAsk (Set Method)

Sets new value for "ColorLineAsk" property.

```
bool ColorLineAsk(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color for Ask line.

Return Value

true - successful, false - cannot change the color.

ColorLineLast (Get Method)

Gets the value of "ColorLineLast" property (color of the last deal price line).

```
color ColorLineLast() const
```

Return Value

Value of "ColorLineLast" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorLineLast (Set Method)

Sets new value for "ColorLineLast" property.

```
bool ColorLineLast(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color of the last deal price line.

Return Value

true - successful, false - cannot change the color.

ColorStopLevels (Get Method)

Gets the value of "ColorStopLevels" property (color of the SL and TP levels).

```
color ColorStopLevels() const
```

Return Value

Value of "ColorStopLevels" property of the chart assigned to the class instance. If there is no chart assigned, it returns [CLR_NONE](#).

ColorStopLevels (Set Method)

Sets new value for "ColorStopLevels" property.

```
bool ColorStopLevels(  
    color new_color    // color  
)
```

Parameters

new_color

[in] New color of the Stop Loss and Take Profit price levels.

Return Value

true - successful, false - cannot change the color.

VisibleBars

Gets total number of visible chart bars.

```
int VisibleBars() const
```

Return Value

Gets total number of visible bars of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

WindowsTotal

Gets total number of chart windows, including the chart indicator subwindows.

```
int WindowsTotal() const
```

Return Value

Total number of windows, including the chart indicator subwindows, assigned to the class instance.
If there is no chart assigned, it returns 0.

WindowIsVisible

Gets visibility flag of the specified chart subwindow.

```
bool WindowIsVisible(  
    int num      // subwindow  
) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Returns visibility flag of the specified chart subwindow assigned to the chart instance. If there is no chart assigned, it returns false.

WindowHandle

Gets window handle of the chart (HWND).

```
int WindowHandle() const
```

Return Value

Window handle of the chart assigned to the chart instance. If there is no chart assigned, it returns [INVALID_HANDLE](#).

FirstVisibleBar

Gets the number of the first visible bar of the chart.

```
int FirstVisibleBar() const
```

Return Value

Number of the first visible bar of the chart assigned to the class instance. If there is no chart assigned, it returns -1.

WidthInBars

Gets chart width in bars.

```
int WidthInBars() const
```

Return Value

Width in chart bars of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

WidthInPixels

Gets chart width in pixels.

```
int WidthInPixels() const
```

Return Value

Width in pixels of the chart assigned to the chart instance. If there is no chart assigned, it returns 0.

HeightInPixels

Gets window height in pixels.

```
int HeightInPixels(  
    int num    // subwindow  
) const
```

Parameters

num

[in] Checked subwindow number (0 means main window).

Return Value

Window height in pixels of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

PriceMin

Gets window minimal price.

```
double PriceMin(  
    int num      // subwindow  
) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Window minimal price value of the chart assigned to the class instance. If there is not chart assigned, it returns [EMPTY_VALUE](#).

PriceMax

Gets window maximal price.

```
double PriceMax(  
    int num      // subwindow  
    ) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Window maximal price value of the chart assigned to the class instance. If there is not chart assigned, it returns [EMPTY_VALUE](#).

Attach

Assigns the current chart to the class instance.

```
void Attach()
```

Attach

Assigns the specified chart to the class instance.

```
void Attach(  
    long chart    // chart identifier  
)
```

Parameters

chart

[in] Identifier of the assigned chart.

FirstChart

Assigns the first chart of the client terminal to the class instance.

```
void FirstChart()
```


NextChart

Assigns the next chart (following the already assigned one) to the class instance.

```
void NextChart()
```


Open

Opens chart with specified parameters and assigns it to the class instance.

```
long Open(  
    const string      symbol_name,      // symbol  
    ENUM_TIMEFRAMES  timeframe         // period  
)
```

Parameters

symbol_name

[in] Chart symbol. [NULL](#) means the symbol of the current chart (to which an expert is attached).

timeframe

[in] Chart timeframe (from [ENUM_TIMEFRAMES](#) enumeration). 0 means the current timeframe.

Return Value

chart identifier.

Detach

Detaches chart from the class instance.

```
void Detach()
```


Close

Closes chart assigned to the class instance.

```
void Close()
```


BringToTop

Show chart on top of other charts.

```
bool BringToTop() const
```

Return Value

true - successful, false - error.

EventObjectCreate

Sets a flag to send notifications of [events](#) of a graphical object creation.

```
bool EventObjectCreate(  
    bool flag    // flag  
)
```

Parameters

flag

[in] New value of a flag to send notifications of events of a graphical object creation.

Return Value

true - successful, false - cannot change the flag.

EventObjectDelete

Sets a flag to send notifications of [events](#) of a graphical object deletion.

```
bool EventObjectDelete(  
    bool flag    // flag  
)
```

Parameters

flag

[in] New value of a flag to send notifications of events of a graphical object deletion.

Return Value

true - successful, false - cannot change the flag.

IndicatorAdd

Adds an indicator with the specified handle into a specified chart window.

```
bool IndicatorAdd(  
    int    sub_win      // number of the subwindow  
    int    handle       // handle of the indicator  
);
```

Parameters

sub_win

[in] The number of the chart subwindow. 0 means the main chart window. if the number of a non-existing window is specified, a new window will be created.

handle

[in] The handle of the indicator.

Return Value

The function returns true in case of success, otherwise it returns false. In order to obtain information about the [error](#), call the [GetLastError\(\)](#) function.

See also

[IndicatorDelete\(\)](#), [IndicatorsTotal\(\)](#), [IndicatorName\(\)](#).

IndicatorDelete

Removes an indicator with a specified name from the specified chart window.

```
bool IndicatorDelete(  
    int          sub_win      // number of the subwindow  
    const string name        // short name of the indicator  
);
```

Parameters

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const name

[in] The short name of the indicator which is set in the [INDICATOR_SHORTNAME](#) property with the [IndicatorSetString\(\)](#) function. To get the short name of an indicator, use the [IndicatorName\(\)](#) function.

Return Value

Returns true in case of successful deletion of the indicator. Otherwise it returns false. To get [error](#) details, use the [GetLastError\(\)](#) function.

Note

If two indicators with identical short names exist in the chart subwindow, the first one in a row will be deleted.

If other indicators on this chart are based on the values of the indicator that is being deleted, such indicators will also be deleted.

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions [iCustom\(\)](#) and [IndicatorCreate\(\)](#). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

Deletion of an indicator from a chart does not mean that its calculation part will be deleted from the terminal memory. To release the indicator handle, use the [IndicatorRelease\(\)](#) function.

The indicator's short name should be formed correctly. It will be written to the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the [IndicatorDelete\(\)](#) function is identified by the short name.

See also

[IndicatorAdd\(\)](#), [IndicatorsTotal\(\)](#), [IndicatorName\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#).

IndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int IndicatorsTotal(  
    long   chart_id, // chart identifier  
    int    sub_win   // number of the subwindow  
);
```

Parameters

chart_id

[in] Chart identifier. 0 denotes the main chart.

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart window.

Return Value

The number of indicators in the specified chart window. To get [error](#) details, use the [GetLastError\(\)](#) function.

Note

The function allows going searching through all the indicators attached to the chart. The number of all the windows of the chart can be obtained from the [CHART_WINDOWS_TOTAL](#) property using the [GetInteger\(\)](#) function.

See also

[IndicatorAdd\(\)](#), [IndicatorDelete\(\)](#), [IndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#).

IndicatorName

Returns the short name of the indicator by the index in the indicators list on the specified chart window.

```
string IndicatorName(  
    int    sub_win    // number of the subwindow  
    int    index       // index of the indicator in the list of indicators added to the  
);
```

Parameters

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart window.

index

[in] Index of the indicator in the list of indicators. The numeration of indicators start with zero, i.e. the first indicator in the list has the 0 index. To obtain the number of indicators in the list, use the [IndicatorsTotal\(\)](#) function.

Return Value

The short name of the indicator which is set in the [INDICATOR_SHORTNAME](#) property with the [IndicatorSetString\(\)](#) function. To get [error](#) details, use the [GetLastError\(\)](#) function.

Note

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions [iCustom\(\)](#) and [IndicatorCreate\(\)](#). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the [INDICATOR_SHORTNAME](#) property using the [IndicatorSetString\(\)](#) function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the [IndicatorDelete\(\)](#) function is identified by the short name.

See also

[IndicatorAdd\(\)](#), [IndicatorDelete](#), [IndicatorsTotal](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#).

Navigate

Shifts the chart.

```
bool Navigate(  
    ENUM_CHART_POSITION position,    // position  
    int shift=0                     // shift  
)
```

Parameters

position

[in] Chart position (from [ENUM_CHART_POSITION](#) enumeration), relative to which a shift is performed.

shift=0

[in] Number of bars to shift.

Return Value

true - successful, false - cannot shift the chart.

Symbol

Gets chart symbol name.

```
string Symbol() const
```

Return Value

Symbol name of the chart, assigned to the class instance. If there is no chart assigned, it returns "".

Period

Gets period of the chart.

```
ENUM_TIMEFRAMES Period() const
```

Return Value

Period of the chart (from [ENUM_TIMEFRAMES](#)) assigned to the class instance. If there is no chart assigned, it returns 0.

Redraw

Redraws chart assigned to the class instance.

```
void Redraw()
```


GetInteger

The function returns the value of the corresponding chart property. The chart property should be of the [integer](#) type. There are two variants of the function.

1. Immediately returns the property value.

```
long GetInteger(  
    ENUM_CHART_PROPERTY_INTEGER prop_id,          // property identifier  
    int sub_window=0                             // subwindow number  
) const
```

2. If successful, puts the value of property to the specified variable of integer type, passed by reference as last parameter.

```
bool GetInteger(  
    ENUM_CHART_PROPERTY_INTEGER prop_id,          // property identifier  
    int sub_window,                             // subwindow number  
    long& value                                  // link to the variable  
) const
```

Parameters

prop_id

[in] Property identifier ([ENUM_CHART_PROPERTY_INTEGER](#) enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Link to the variable that receives the value of the requested property.

Return Value

Value of property of the chart assigned to the class instance. If there is not any chart assigned, it returns -1.

For the second variant, the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise it returns false. To read more about the [error](#), call [GetLastError\(\)](#).

SetInteger

Sets new value for the property of the integer type.

```
bool SetInteger(  
    ENUM_CHART_PROPERTY_INTEGER prop_id,    // property identifier  
    long value                               // value  
)
```

Parameters

prop_id

[in] Chart property identifier (from [ENUM_CHART_PROPERTY_INTEGER](#) enumeration).

value

[in] New value of the property.

Return Value

true - successful, false - cannot change the integer property.

GetDouble

The function returns the value of the corresponding chart property. The object property should be of the double type. There are two variants of the function.

1. Immediately returns the property value.

```
double GetDouble (
    ENUM_CHART_PROPERTY_DOUBLE prop_id,      // property identifier
    int sub_window=0                        // subwindow number
) const
```

2. If successful, puts the value of property to the specified variable of double type, passed by reference as last parameter.

```
bool GetDouble (
    ENUM_CHART_PROPERTY_DOUBLE prop_id,      // property identifier
    int sub_window,                        // subwindow number
    double& value                          // link to the variable
) const
```

Parameters

prop_id

[in] Chart property identifier (from [ENUM_CHART_PROPERTY_DOUBLE](#) enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Variable of the double type that received the value of the requested property.

Return Value

Value of property of the chart assigned to the class instance. If there is not any chart assigned, it returns [EMPTY_VALUE](#).

For the second variant the function, it returns true if the property value is received, otherwise returns false. To read more about the [error](#), call [GetLastError\(\)](#).

SetDouble

Sets new value for the chart property of the double type.

```
bool SetDouble(  
    ENUM_CHART_PROPERTY_DOUBLE prop_id,    // property identifier  
    double value                          // new value  
)
```

Parameters

prop_id

[in] Chart property identifier (from [ENUM_CHART_PROPERTY_DOUBLE](#) enumeration).

value

[in] New value for the property.

Return Value

true - successful, false - cannot change the double property.

GetString

The function returns the value of the corresponding chart property. The chart property should be of the string type. There are two variants of the function.

1. Immediately returns the property value.

```
string GetString(  
    ENUM_CHART_PROPERTY_STRING prop_id    // property identifier  
) const
```

2. If successful, puts the value of property to the specified variable of string type, passed by reference as last parameter.

```
bool GetString(  
    ENUM_CHART_PROPERTY_STRING prop_id,    // property identifier  
    string& value                          // link to the variable  
) const
```

Parameters

prop_id

[in] Chart property identifier (from [ENUM_CHART_PROPERTY_STRING](#) enumeration).

value

[in] Link to the variable that receives the value of the requested property.

Return Value

Value of a chart property assigned to the class instance. If there is no chart assigned, it returns "".

For the second variant, the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise it returns false. To read more about the [error](#), call [GetLastError\(\)](#).

SetString

Sets new value for the chart property of the string type.

```
bool SetString(  
    ENUM_CHART_PROPERTY_STRING prop_id,    // property identifier  
    string value                          // value  
)
```

Parameters

prop_id

[in] Chart property identifier (from [ENUM_CHART_PROPERTY_STRING](#) enumeration).

value

[in] New value for the property.

Return Value

true - successful, false - cannot change the string property.

SetSymbolPeriod

Changes symbol and period of the chart assigned to the class instance.

```
bool SetSymbolPeriod(  
    const string      symbol_name,      // symbol  
    ENUM_TIMEFRAMES timeframe          // period  
)
```

Parameters

symbol_name

[in] New chart symbol. [NULL](#) means the symbol of the current chart (to which an expert is attached).

timeframe

[in] New chart timeframe (from [ENUM_TIMEFRAMES](#) enumeration). 0 means the current chart timeframe.

Return Value

true - successful, false - cannot change the property.

ApplyTemplate

Applies specified template to the chart.

```
bool ApplyTemplate(  
    const string filename    // template  
)
```

Parameters

filename

[in] File name of the template.

Return Value

true - successful, false - cannot apply the template.

ScreenShot

Creates a screenshot of the specified chart in its current state in .gif format.

```
bool  ScreenShot(  
    string      filename,           // file name  
    int         width,              // width  
    int         height,             // height  
    ENUM_ALIGN_MODE align_mode=ALIGN_RIGHT // align type  
) const
```

Parameters

filename

[in] File name for screenshot.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Align mode, if screenshot is narrow.

Return Value

true - successful, false - error.

WindowOnDropped

Gets chart subwindow number corresponding to the object (expert or script) drop point.

```
int WindowOnDropped() const
```

Return Value

Chart subwindow number of the object drop point. 0 means main chart window.

PriceOnDropped

Gets price coordinate corresponding to the object (expert or script) drop point.

```
double PriceOnDropped() const
```

Return Value

Price coordinate of the object drop point.

TimeOnDropped

Gets time coordinate corresponding to the object (expert or script) drop point.

```
datetime TimeOnDropped() const
```

Return Value

Time coordinate of the object drop point.

XOnDropped

Gets X coordinate corresponding to the object (expert or script) drop point.

```
int XOnDropped() const
```

Return Value

X coordinate of the object drop point.

YOnDropped

Gets Y coordinate corresponding to the object (expert or script) drop point.

```
int YOnDropped() const
```

Return Value

Y coordinate of the object drop point.

Save

Saves object parameters to file.

```
virtual bool Save(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen\(...\)](#) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool Load(  
    int file_handle    // file handle  
)
```

Parameters

file_handle

[in] handle of the binary file already opened by [FileOpen\(...\)](#) function.

Return Value

true - successful, false - error.

Type

Returns type identifier.

```
virtual int Type() const
```

Return Value

Type identifier (0x1111 for CChart).

Graphics

The graphics library contains classes and global functions for quick plotting of custom charts.

The library provides convenient ready-made solutions for building axes, curves, as well as the methods for quick access to changing common properties of a custom chart.

The graphics library is placed to the Include\Graphics folder of the terminal's working directory.

Class	Description
<u>CAxis</u>	Class for working with coordinate axes
<u>ColorGenerator</u>	Class specifying the default color scheme
<u>CCurve</u>	Class for working with curves
<u>CGraphic</u>	Base class for creating custom charts

GraphPlot

Functions for quick curve plotting.

Version for plotting a single curve using Y coordinates.

```
string GraphPlot(  
    const double    &y[],           // Y coordinates  
    ENUM_CURVE_TYPE type=CURVE_POINTS // curve type  
)
```

Note

Y array indices are used as X coordinates for the curve.

Version for plotting a single curve using X and Y coordinates

```
string GraphPlot(  
    const double    &x[],           // X coordinates  
    const double    &y[],           // Y coordinates  
    ENUM_CURVE_TYPE type=CURVE_POINTS // curve type  
)
```

Version for plotting two curves using X and Y coordinates

```
string GraphPlot(  
    const double    &x1[],           // X coordinates  
    const double    &y1[],           // Y coordinates  
    const double    &x2[],           // X coordinates  
    const double    &y2[],           // Y coordinates  
    ENUM_CURVE_TYPE type=CURVE_POINTS // curve type  
)
```

Version for plotting three curves using X and Y coordinates

```
string GraphPlot(  
    const double    &x1[],           // X coordinates  
    const double    &y1[],           // Y coordinates  
    const double    &x2[],           // X coordinates  
    const double    &y2[],           // Y coordinates  
    const double    &x3[],           // X coordinates  
    const double    &y3[],           // Y coordinates  
    ENUM_CURVE_TYPE type=CURVE_POINTS // curve type  
)
```


Version for plotting a curve using CPoint2D points coordinates

```

string  GraphPlot(
    const CPoint2D    &points[],           // curve coordinates
    ENUM_CURVE_TYPE  type=CURVE_POINTS    // curve type
)

```

Version for plotting two curves using CPoint2D points coordinates

```

string  GraphPlot(
    const CPoint2D    &points1[],          // curve coordinates
    const CPoint2D    &points2[],          // curve coordinates
    ENUM_CURVE_TYPE  type=CURVE_POINTS    // curve type
)

```

Version for plotting three curves using CPoint2D points coordinates

```

string  GraphPlot(
    const CPoint2D    &points1[],          // curve coordinates
    const CPoint2D    &points2[],          // curve coordinates
    const CPoint2D    &points3[],          // curve coordinates
    ENUM_CURVE_TYPE  type=CURVE_POINTS    // curve type
)

```

Version for plotting a curve using the pointer to CurveFunction

```

string  GraphPlot(
    CurveFunction      function,            // pointer to the function
    const double       from,               // initial value of the argument
    const double       to,                 // final value of the argument
    const double       step,               // increment by the argument
    ENUM_CURVE_TYPE    type=CURVE_POINTS   // curve type
)

```

Version for plotting two curves using the pointers to the CurveFunction functions

```

string  GraphPlot(
    CurveFunction      function1,          // pointer to the function
    CurveFunction      function2,          // pointer to the function
    const double       from,               // initial value of the argument
    const double       to,                 // final value of the argument
    const double       step,               // increment by the argument
    ENUM_CURVE_TYPE    type=CURVE_POINTS   // curve type
)

```


Version for plotting three curves using the pointers to the CurveFunction functions

```

string  GraphPlot (
    CurveFunction  function1,           // pointer to the function
    CurveFunction  function2,           // pointer to the function
    CurveFunction  function3,           // pointer to the function
    const double   from,                // initial value of the argument
    const double   to,                  // final value of the argument
    const double   step,                // increment by the argument
    ENUM_CURVE_TYPE type=CURVE_POINTS // curve type
)

```

Parameters

&x[]

[in] X coordinates.

&y[]

[in] Y coordinates.

&x1[]

[in] X coordinates for the first curve.

&y1[]

[in] Y coordinates for the first curve.

&x2[]

[in] X coordinates for the second curve.

&y2[]

[in] Y coordinates for the second curve.

&x3[]

[in] X coordinates for the third curve.

&y3[]

[in] Y coordinates for the third curve.

&points[]

[in] Coordinates of the curve dots.

&points1[]

[in] Coordinates of the first curve dots.

&points2[]

[in] Coordinates of the second curve dots.

&points3[]

[in] Coordinates of the third curve dots.

function

[in] Pointer to the CurveFunction function.

function1

[in] Pointer to the first function.

function2

[in] Pointer to the second function.

function3

[in] Pointer to the third function.

from

[in] Corresponds to the first X coordinate.

to

[in] Corresponds to the last X coordinate.

step

[in] Parameter for calculating the X coordinates.

type=CURVE_POINTS

[in] Curve type.

Return Value

Name of a graphical resource.

CAxis

CAxis is an auxiliary graphics library class for working with the coordinate axes.

Description

The CAxis class receives and stores various parameters of the coordinate axes. The class implements the ability to auto scale the coordinate axes dynamically.

Declaration

```
class CAxis
```

Title

```
#include <Graphics\Axis.mqh>
```

Class methods

Method	Description
AutoScale	Get/set the auto-scaling flag
Min	Get/set the minimum axis value
Max	Get/set the maximum axis value
Step	Get the step value by axis
Name	Get/set the axis name
Color	Get/set the axis color
ValuesSize	Get/set the size of the axis numbers
ValuesWidth	Get/set the maximum displayed length of the axis numbers
ValuesFormat	Get/set the format of the axis numbers
ValuesDateTimeMode	Get the format of converting a date into a string.
ValuesFunctionFormat	Get the pointer to the function defining the format of displaying values on the axis.
ValuesFunctionFormatCBData	Get the pointer to the object that may contain additional data on converting axis values.
NameSize	Get/set the font size of the axis name
ZeroLever	Get/set the "zero lever" value
DefaultStep	Get/set the initial step value by axis
MaxLabels	Get/set the maximum amount of numbers on the axis

<u>MinGrace</u>	Get/set the "tolerance" value for the axis minimum
<u>MaxGrace</u>	Get/set the "tolerance" value for the axis maximum
<u>SelectAxisScale</u>	Auto scale the axis.

AutoScale (Get method)

Returns the flag defining the need for auto-scale.

```
bool AutoScale()
```

Return Value

The flag value.

Note

true — perform auto-scaling.

false — do not perform auto-scaling.

AutoScale (Set method)

Sets the flag defining the need for auto-scale.

```
void AutoScale (  
    const bool auto    // flag value  
)
```

Parameters

auto

[in]

Note

true — perform auto-scaling.

false — do not perform auto-scaling.

Min (Get method)

Returns the minimum axis value.

```
double Min()
```

Return Value

Minimum axis value.

Min (Set method)

Sets the minimum axis value.

```
void Min(  
    const double min    // minimum value  
)
```

Parameters

min

[in] Minimum value.

Max (Get method)

Returns the maximum axis value.

```
double Max()
```

Return Value

Maximum axis value.

Max (Set method)

Sets the maximum axis value.

```
void Max(  
    const double max // maximum value  
)
```

Parameters

max

[in] Maximum axis value.

Step (Get method)

Return the step value by axis.

```
double Step()
```

Return Value

Step value.

Name (Get method)

Returns the axis name.

```
string Name()
```

Return Value

Axis name.

Name (Set method)

Sets the axis name.

```
void Name(  
    const string name    // axis name  
)
```

Parameters

name

[in] Axis name.

Color (Get method)

Returns the axis color.

```
color Color()
```

Return Value

Axis color.

Color (Set method)

Sets the axis color.

```
void Color(  
    const color clr    // axis color  
)
```

Parameters

clr

[in] Axis color.

ValuesSize (Get method)

Returns the size of the axis numbers.

```
int ValuesSize()
```

Return Value

Size of the axis numbers.

ValuesSize (Set method)

Sets the size of the axis numbers.

```
void ValuesSize(  
    const int size    // size of the axis numbers  
)
```

Parameters

size

[in] Size of the axis numbers

ValuesWidth (Get method)

Returns the maximum allowed length in pixels for displaying the axis numbers.

```
int ValuesWidth()
```

Return Value

Length of the axis numbers in pixels.

Note

If a length in pixels for a specified number exceeds the maximum allowed display length, it is truncated and ends in dots.

ValuesWidth (Set method)

Sets the maximum allowed length in pixels for displaying the axis numbers.

```
void ValuesWidth(  
    const int width // maximum allowed length in pixels  
)
```

Parameters

width

[in] Maximum allowed length of the axis numbers.

Note

If a length in pixels for a specified number exceeds the maximum allowed display length, it is truncated and ends in dots.

ValuesFormat (Get method)

Returns the format of the axis numbers.

```
string ValuesFormat()
```

Return Value

Number format.

ValuesFormat (Set method)

Sets the format of the axis numbers.

```
void ValuesFormat(  
    const string format      // format of the axis numbers  
)
```

Parameters

format

[in] Format of the axis numbers.

ValuesDateTimeMode (Get method)

Get the format of converting a date into a string.

```
int ValuesDateTimeMode()
```

Return Value

Format of converting a date into a string.

ValuesDateTimeMode (Set method)

Set the format of converting a date into a string.

```
void ValuesDateTimeMode(  
    const int mode // format of converting a date into a string  
)
```

Parameters

mode

[in] Conversion format.

Note

Find out more about the formats of converting a date into a string in the [TimeToString\(\)](#) function description.

ValuesFunctionFormat (Get method)

Get the pointer to the function defining the format of displaying values on the axis.

```
DoubleToStringFunction ValuesFunctionFormat ()
```

Return Value

Pointer to the function defining the format of displaying values on the axis.

ValuesFunctionFormat (Set method)

Set the pointer to the function defining the format of displaying values on the axis.

```
void ValuesFunctionFormat (  
    DoubleToStringFunction func    // function for converting numerical values into  
)
```

Parameters

func

[in] Custom function for converting numerical values into a string.

Example:



The format of displaying X axis values has been changed using the following code:


```

//+-----+
//|                                     DateAxisGraphic.mqh |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//--- array for store values
double arrX[];
double arrY[];
//+-----+
//| Custom function for create values on X-axis |
//+-----+
string TimeFormat(double x,void *cbdata)
{
    return(TimeToString((datetime)arrX[ArraySize(arrX)-(int)x-1]));
}
//+-----+
void OnStart()
{
    MqlRates rates[];
    CopyRates(Symbol(),Period(),0,100,rates);
    ArraySetAsSeries(rates,true);
    int size=ArraySize(rates);
    ArrayResize(arrX,size);
    ArrayResize(arrY,size);
    for(int i=0; i<size;++i)
    {
        arrX[i]=(double)rates[i].time;
        arrY[i]=rates[i].close;
    }
    //--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0,"DateAxisGraphic",0,30,30,780,380))
    {
        graphic.Attach(0,"DateAxisGraphic");
    }
    //--- create curve
    CCurve *curve=graphic.CurveAdd(arrY,CURVE_LINES);
    //--- gets the X-axis
    CAxis *xAxis=graphic.XAxis();
    //--- sets the X-axis properties
    xAxis.AutoScale(false);
    xAxis.Type(AXIS_TYPE_CUSTOM);
    xAxis.ValuesFunctionFormat(TimeFormat);
    xAxis.DefaultStep(20.0);
    //--- plot
    graphic.CurvePlotAll();
    graphic.Update();
}

```


ValuesFunctionFormatCBData (Get method)

Get the pointer to the object that may contain additional data on converting axis values.

```
void* ValuesFunctionFormatCBData()
```

Return Value

The pointer to the object that may contain additional data on converting axis values.

ValuesFunctionFormatCBData (Set method)

Set the pointer to the class object that may contain additional data on converting axis values.

```
void ValuesFunctionFormatCBData(  
    void* cbdata    // pointer to the class object  
)
```

Parameters

cbdata

[in] Pointer to any class object containing additional data on converting axis values

NameSize (Get method)

Returns the font size of the axis name.

```
int NameSize()
```

Return Value

Font size of the axis name.

NameSize (Set method)

Sets the font size of the axis name.

```
void NameSize(  
    const int size    // font size of the axis name  
)
```

Parameters

size

[in] Font size of the axis name.

ZeroLever (Get method)

Returns the "zero lever" value.

```
double ZeroLever()
```

Return Value

"Zero lever".

Note

The value is used to define when the axis scale range should be expanded to include a zero value.

ZeroLever (Set method)

Sets the "zero lever" value.

```
void ZeroLever(  
    const double value    // "zero lever" value  
)
```

Parameters

value

[in] "Zero lever" value.

Note

The value is used to define when the axis scale range should be expanded to include a zero value.

DefaultStep (Get method)

Returns the initial step value by axis

```
double DefaultStep()
```

Return Value

Step by axis.

DefaultStep (Set method)

Sets the initial step value by axis

```
void DefaultStep(  
    const double value    // step by axis  
)
```

Parameters

value

[in] Initial step value by axis.

MaxLabels (Get method)

Returns the maximum allowed amount of numbers displayed on the axis.

```
double MaxLabels()
```

Return Value

Maximum amount of numbers on the axis.

MaxLabels (Set method)

Sets the maximum allowed amount of numbers displayed on the axis.

```
void MaxLabels(  
    const double value // maximum number  
)
```

Parameters

value

[in] Maximum allowed amount of numbers displayed on the axis

MinGrace (Get method)

Returns the "tolerance" applied to the axis minimum.

```
double MinGrace()
```

Return Value

"Tolerance" value for the axis minimum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

MinGrace (Set method)

Sets the "tolerance" applied to the axis minimum.

```
void MinGrace(  
    const double value    // "tolerance" value  
)
```

Parameters

value

[in] "Tolerance" applied to the axis minimum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

MaxGrace (Get method)

Returns the "tolerance" applied to the axis maximum.

```
double MaxGrace()
```

Return Value

"Tolerance" value for the axis maximum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MaxGrace is equal to 0.1, then 10% of the axis length (or 1.2) is added to the maximum value. As a result, the axis covers the interval from 4.0 to 17.2.

MaxGrace (Set method)

Sets the "tolerance" applied to the axis maximum.

```
void MaxGrace(  
    const double value    // "tolerance" value  
)
```

Parameters

value

[in] "Tolerance" value applied to the axis maximum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

SelectAxisScale

Auto scale the axis.

```
void SelectAxisScale()
```


CColorGenerator

CColorGenerator class is an auxiliary graphics library class for working with the color palette.

Description

The CColorGenerator class contains the initial color palette used for curves by default (if a color is not specified by a user).

If all colors from the initial palette are used already, new colors are automatically generated and the palette is refilled.

Declaration

```
class CColorGenerator
```

Title

```
#include <Graphics\ColorGenerator.mqh>
```

Class methods

Method	Description
Next	Returns the next color from the palette
Reset	Resets the generator

Next

Returns the next color from the palette.

```
uint Next ()
```

Return Value

Color.

Note

If all colors from the palette have already been passed through, new colors are automatically generated in order to replace the old ones in the palette.

Reset

Resets the generator.

```
void Reset()
```


CCurve

The CCurve class works with the properties of the curves generated on the chart.

Description

The CCurve class sets, installs and receives the coordinates and various properties of the curves when working with the CGraphic class.

There are three curve plotting modes: dots, lines and histogram. Separate parameters are implemented for each plotting mode in the class.

Declaration

```
class CCurve : public CObject
```

Title

```
#include <Graphics\Curve.mqh>
```

Inheritance hierarchy

[CObject](#)

CCurve

Class methods

Method	Description
Type	Get the curve type
Name	Get the curve name
Color	Get the curve color
XMax	Get the maximum value of the X function
XMin	Get the minimum value of the X function
YMax	Get the maximum value of the Y function
YMin	Get the minimum value of the Y function
Size	Get the number of dots defining a curve
PointsSize	Get/set the linear size of dots defining a curve
PointsFill	Get/set the flag for filling dots defining a curve
PointsColor	Get/set the dot filling color
GetX	Gets X values of all curve dots to the array
GetY	Gets Y values of all curve dots to the array

<u>LineStyle</u>	Get/set a line style when plotting a curve using lines
<u>LinesIsSmooth</u>	Get/set the smoothing flag when drawing using lines
<u>LinesSmoothTension</u>	Get/set the curve smoothing parameter when drawing using lines
<u>LinesSmoothStep</u>	Get/set the length of the approximating lines for smoothing when plotting by lines
<u>LinesWidth</u>	Get/set a line width when plotting a curve using lines
<u>HistogramWidth</u>	Get/set the width of columns when plotting using a histogram
<u>CustomPlotCBData</u>	Get/set the pointer to the object to be used in the custom curve plotting mode.
<u>CustomPlotFunction</u>	Get/set the pointer to the function implementing the custom curve plotting mode.
<u>PointsType</u>	Get/set the flag pointing at the type of dots used when plotting a dotted curve.
<u>StepsDimension</u>	Get/set the value indicating the dimension used in step-type curve rendering.
<u>TrendLineCoefficients</u>	Get/set trend line ratios for writing them into an array.
<u>TrendLineColor</u>	Get/set a color of a trend line for a curve.
<u>TrendLineVisible</u>	Get/set the trend line visibility flag.
<u>Update</u>	Update the curve coordinates.
<u>Visible</u>	Get/set the flag defining if a function is visible on the chart.

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Compare](#)

Type

Return the curve type.

```
ENUM_CURVE_TYPE Type ()
```

Return Value

Curve type.

Name

Returns the curve name.

```
string Name()
```

Return Value

Curve name.

Color

Returns the curve color.

```
uint Color()
```

Return Value

Curve color.

XMax

Returns the maximum value of the X function (real numbers only).

```
double XMax()
```

Return Value

Maximum real number among all the function arguments.

XMin

Returns the minimum value of the X function (real numbers only).

```
double XMin()
```

Return Value

Minimum real number among all the function arguments.

YMax

Returns the maximum value of the Y function (real numbers only).

```
double YMax()
```

Return Value

Maximum value of the Y function (real numbers only).

YMin

Returns the minimum value of the Y function (real numbers only).

```
double YMin()
```

Return Value

Minimum value of the Y function (real numbers only).

Size

Returns the number of dots defining the curve.

```
int Size()
```

Return Value

Number of dots defining the curve.

PointSize (Get method)

Returns the linear size (in pixels) of dots used in plotting the curve.

```
int PointSize()
```

Return Value

Size of dots defining the curve in pixels.

PointSize (Set method)

Sets the linear size (in pixels) of dots used in plotting the curve.

```
void PointSize(  
    const int size    // dot size in pixels  
)
```

Parameters

size

[in] Linear size (in pixels) of dots used in plotting the curve.

PointsFill (Get method)

Returns a flag determining if a filling for dots defining a curve should be performed.

```
bool PointsFill ()
```

Return Value

The flag value.

Note

true — perform a filling

false — do not perform a filling

PointsFill (Set method)

Sets a flag determining if a filling for dots defining a curve should be performed.

```
void PointsFill (  
    const bool fill    // flag value  
)
```

Parameters

fill

[in] Flag value.

Note

true — perform a filling

false — do not perform a filling

PointsColor (Get method)

Returns the dot filling color.

```
uint PointsColor ()
```

Return Value

Color of filling dots defining the curve.

PointsColor (Set method)

Sets the dot filling color

```
void PointsColor (  
    const uint clr      //dot filling color  
)
```

Parameters

clr

[in] Color of filling dots defining the curve.

GetX

Gets X values of all curve dots to the array.

```
void GetX(  
    double& x[]    // array for writing X values  
)
```

Parameters

`x[]`

[out] Array for getting X values of all curve dots.

Note

Each curve dot is defined by a couple of X and Y values. These values are not coordinates in pixels for drawing in the [CGraphic](#) class.

GetY

Gets Y values of all curve dots to the array.

```
void GetY(  
    double& y[]    // array for writing Y values  
)
```

Parameters

y[]

[out] Array for getting Y values of all curve dots.

Note

Each curve dot is defined by a couple of X and Y values. These values are not coordinates in pixels for drawing in the [CGraphic](#) class.

LineStyle (Get method)

Returns a line style when plotting a curve using lines.

```
ENUM_LINE_STYLE LineStyle()
```

Return Value

Line style.

LineStyle (Set method)

Sets a line style when plotting a curve using lines.

```
void LineStyle (  
    ENUM_LINE_STYLE style    // line style  
)
```

Parameters

style

[in] Line style.

LinesIsSmooth (Get method)

Returns a flag defining if smoothing should be done when plotting a curve by lines.

```
bool LinesIsSmooth()
```

Return Value

Flag value

Note

true — perform smoothing

false — do not perform smoothing

LinesIsSmooth (Set method)

Sets a flag defining if smoothing should be done when plotting a curve by lines.

```
void LinesIsSmooth(  
    const bool smooth // flag value  
)
```

Parameters

smooth

[in] Flag value

Note

true — perform smoothing

false — do not perform smoothing

LinesSmoothTension (Get method)

Returns the curve smoothing parameter when drawing using lines.

```
double LinesSmoothTension()
```

Return Value

Smoothing parameter value

Note

The 'tension' value is within the (0.0; 1.0] range.

LinesSmoothTension (Set method)

Sets the curve smoothing parameter when drawing using lines.

```
void LinesSmoothTension (  
    const double tension    // parameter value  
)
```

Parameters

tension

[in] Smoothing parameter value.

Note

The 'tension' value is within the (0.0; 1.0] range.

LinesSmoothStep (Get method)

Returns the length of the approximating lines for smoothing when plotting by lines.

```
double LinesSmoothStep()
```

Return Value

Length of approximating lines in pixels.

LinesSmoothStep (Set method)

Sets the length of the approximating lines for smoothing when plotting by lines.

```
void LinesSmoothStep(  
    const double step    // line length  
)
```

Parameters

step

[in] Length of approximating lines

LinesEndStyle (Set method)

Get the flag indicating [lines end plotting style](#) when using lines to plot a curve.

```
ENUM_LINE_END LinesEndStyle()
```

Return Value

A value of the flag indicating lines end plotting style when using lines to plot a curve.

LinesEndStyle (Get method)

Set the flag indicating lines end plotting style when using lines to plot a curve.

```
void LinesEndStyle(  
    ENUM_LINE_END end_style // flag value  
)
```

Parameters

end_style

[in] A value of the flag indicating line end plotting style when using lines to plot a curve.

LineWidth (Get method)

Get lines width when plotting a curve using lines.

```
int LineWidth()
```

Return Value

Lines width.

LineWidth (Set method)

Set lines width when plotting a curve using lines.

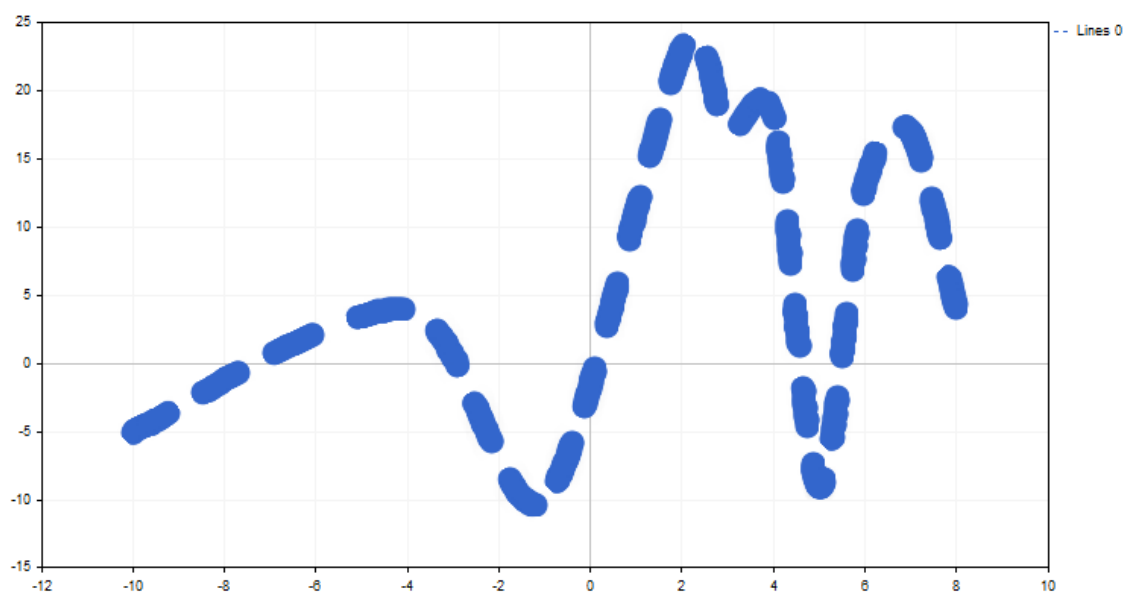
```
void LineWidth(  
    const int width // lines width  
)
```

Parameters

width

[in] Lines width when plotting a curve using lines.

Example:



A line width has been changed using the following code:


```

//+-----+
//|                                     CandleGraphic.mq5 |
//|                                     Copyright 2016, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    double x[] = { -100,-40,-10,20,30,40,50,60,70,80,120 };
    double y[] = { -5,4,-10,23,17,18,-9,13,17,4,9 };
//--- create graphic
CGraphic graphic;
if (!graphic.Create(0,"ThickLineGraphic",0,30,30,780,380))
{
    graphic.Attach(0,"ThickLineGraphic");
}
//--- create curve
CCurve *curve=graphic.CurveAdd(x,y,CURVE_LINES);
//--- sets the curve properties
curve.LinesSmooth(true);
curve.LinesStyle(STYLE_DASH);
curve.LinesEndStyle(LINE_END_ROUND);
curve.LinesWidth(10);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```


HistogramWidth (Get method)

Returns the width of columns when plotting using a histogram.

```
int HistogramWidth()
```

Return Value

Column width in pixels.

HistogramWidth (Set method)

Sets the width of columns when plotting using a histogram.

```
void HistogramWidth(  
    const int width // column width  
)
```

Parameters

width

[in] Column width in pixels.

CustomPlotCBData (Get method)

Get the pointer to the object to be used in the custom curve plotting mode.

```
void* CustomPlotCBData()
```

Return Value

Pointer to the object for the custom curve plotting mode.

CustomPlotCBData (Set method)

Set the pointer to the object to be used in the custom curve plotting mode.

```
void CustomPlotCBData(  
    void* cldata    // pointer to the object  
)
```

Parameters

cldata

[in] The pointer to the object to be used in the custom curve plotting mode

CustomPlotFunction (Get method)

Get the pointer to the function implementing the custom curve plotting mode.

```
PlotFuction CustomPlotFunction()
```

Return Value

Pointer to the function implementing the custom curve plotting mode.

CustomPlotFunction (Set method)

Set the pointer to the function implementing the custom curve plotting mode.

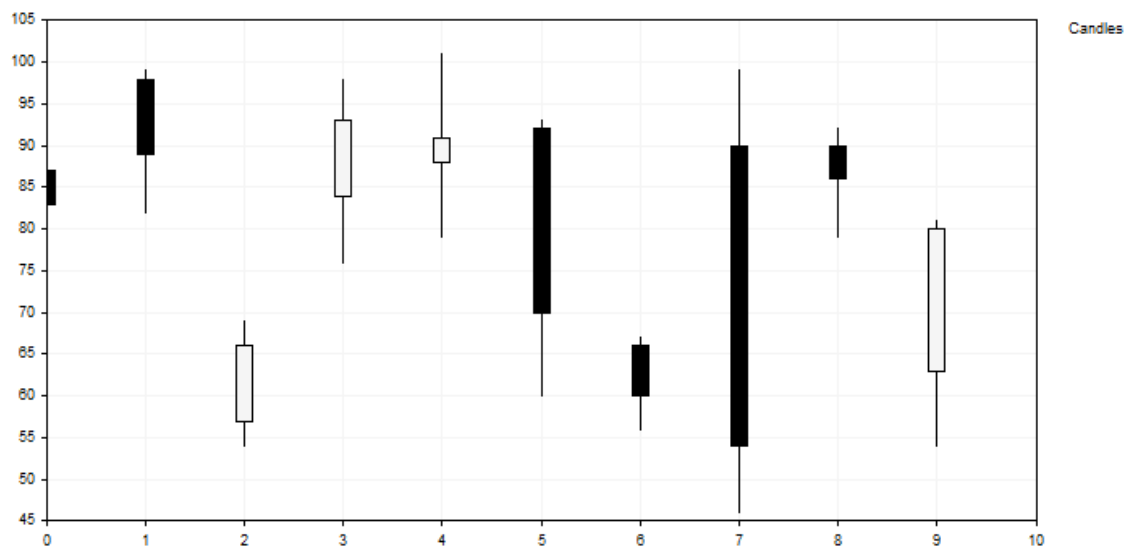
```
void CustomPlotFunction(  
    PlotFuction func    // pointer to the function  
)
```

Parameters

func

[in] Pointer to the function implementing the custom curve plotting mode

Example:



This curve consisting of bars is built using the following code:


```

//+-----+
//|                                     CandleGraphic.mq5 |
//|               Copyright 2016, MetaQuotes Software Corp. |
//|               https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Class CCandle |
//| Usage: class to represent the candle |
//+-----+
class CCandle: public CObject
{
private:
    double      m_open;
    double      m_close;
    double      m_high;
    double      m_low;
    uint        m_clr_inc;
    uint        m_clr_dec;
    int         m_width;

public:
    CCandle(const double open,const double close,const double high,const double low,
            const int width,const uint clr_inc,const uint clr_dec)
    ~CCandle(void);

    double      OpenValue(void) const { return(m_open); }
    double      CloseValue(void) const { return(m_close); }
    double      HighValue(void) const { return(m_high); }
    double      LowValue(void) const { return(m_low); }
    uint        CandleColorIncrement(void) const { return(m_clr_inc); }
    uint        CandleColorDecrement(void) const { return(m_clr_dec); }
    int         CandleWidth(void) const { return(m_width); }
};
//+-----+
//| Constructor |
//+-----+
CCandle::CCandle(const double open,const double close,const double high,const double low,
                const int width,const uint clr_inc=0x000000,const uint clr_dec=0x000000,
                m_open(open),m_close(close),m_high(high),m_low(low),
                m_clr_inc(clr_inc),m_clr_dec(clr_dec),m_width(width)
    {
    }
//+-----+
//| Destructor |
//+-----+
CCandle::~CCandle(void)
{
}
//+-----+
//| Custom method for plot candles |
//+-----+
void PlotCandles(double &x[],double &y[],int size,CGraphic *graphic,CCanvas *canvas,void *obj)
{
    //--- check obj
    CArrayObj *candles=dynamic_cast<CArrayObj*>(cbdata);
    if(candles==NULL || candles.Total()!=size)
        return;
    //--- plot candles
    for(int i=0; i<size; i++)
    {
        CCandle *candle=dynamic_cast<CCandle*>(candles.At(i));
    }
}

```



```

        if(candle==NULL)
            return;
        //--- primary calculate
        int xc=graphic.ScaleX(x[i]);
        int width_2=candle.CandleWidth()/2;
        int open=graphic.ScaleY(candle.OpenValue());
        int close=graphic.ScaleY(candle.CloseValue());
        int high=graphic.ScaleY(candle.HigthValue());
        int low=graphic.ScaleY(candle.LowValue());
        uint clr=(open<=close) ? candle.CandleColorIncrement() : candle.CandleColorDecrement();
        //--- plot candle
        canvas.LineVertical(xc,high,low,0x000000);
        //--- plot candle real body
        canvas.FillRectangle(xc+width_2,open,xc-width_2,close,clr);
        canvas.Rectangle(xc+width_2,open,xc-width_2,close,0x000000);
    }
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    int count=10;
    int width=10;
    double x[];
    double y[];
    ArrayResize(x,count);
    ArrayResize(y,count);
    CArrayObj candles();
    double max=0;
    double min=0;
    //--- create values
    for(int i=0; i<count; i++)
    {
        x[i] = i;
        y[i] = i;
        //--- calculate values
        double open=MathRound(50.0+(MathRand()/32767.0)*50.0);
        double close=MathRound(50.0+(MathRand()/32767.0)*50.0);
        double high=MathRound(MathMax(open,close)+(MathRand()/32767.0)*10.0);
        double low=MathRound(MathMin(open,close)-(MathRand()/32767.0)*10.0);
        //--- find max and min
        if(i==0 || max<high)
            max=high;
        if(i==0 || min>low)
            min=low;
        //--- create candle
        CCandle *candle=new CCandle(open,close,high,low,width);
        candles.Add(candle);
    }
    //--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0,"CandleGraphic",0,30,30,780,380))
    {
        graphic.Attach(0,"CandleGraphic");
    }
    //--- create curve
    CCurve *curve=graphic.CurveAdd(x,y,CURVE_CUSTOM,"Candles");
    //--- sets the curve properties
    curve.CustomPlotFunction(PlotCandles);
    curve.CustomPlotCBData(GetPointer(candles));
}

```



```
//--- sets the graphic properties
    graphic.YAxis().Max((int)max);
    graphic.YAxis().Min((int)min);
//--- plot
    graphic.CurvePlotAll();
    graphic.Update();
}
```


PointsType (Get method)

Get the flag pointing at the type of dots used when plotting a dotted curve.

```
ENUM_POINT_TYPE PointsType()
```

Return Value

A value of the flag indicating a type of dots.

PointsType (Set method)

Set the flag pointing at the type of dots used when plotting a dotted curve.

```
void PointsType(  
    ENUM_POINT_TYPE type    // flag value  
)
```

Parameters

type

[in] A value of the flag pointing at the type of dots used when plotting a dotted curve.

StepsDimension (Get method)

Get the value indicating the dimension used in step-type curve rendering.

```
int StepsDimension()
```

Return Value

Dimension used in step-type curve rendering.

StepsDimension (Set method)

Set the value indicating the dimension used in step-type curve rendering.

```
void StepsDimension(  
    const int dimension    // dimension  
)
```

Parameters

dimension

[in] Dimension (0 or 1).

Note

0 — x (the horizontal line is followed by the vertical one).

1 — y (the vertical line is followed by the horizontal one).

TrendLineCoefficients (Get method)

Get trend line ratios for writing them into an array.

```
double& TrendLineCoefficients ()
```

Return Value

Trend line ratios.

TrendLineCoefficients (Set method)

Set trend line ratios for writing then into an array.

```
void TrendLineCoefficients (  
    double& coefficients[]    // array for writing ratios  
)
```

Parameters

coefficients[]

[out] Array for writing ratios.

TrendLineColor (Get method)

Get a color of a trend line for a curve.

```
uint TrendLineColor()
```

Return Value

Color of the trend line.

TrendLineColor (Set method)

Set a color of a trend line for a curve.

```
void TrendLineColor(  
    const uint clr    // trend line color  
)
```

Parameters

clr

[in] Line color

TrendLineVisible (Get method)

Get the trend line visibility flag.

```
bool TrendLineVisible()
```

Return Value

A value of the flag that specifies if a trend line is visible.

TrendLineVisible (Set method)

Set the trend line visibility flag.

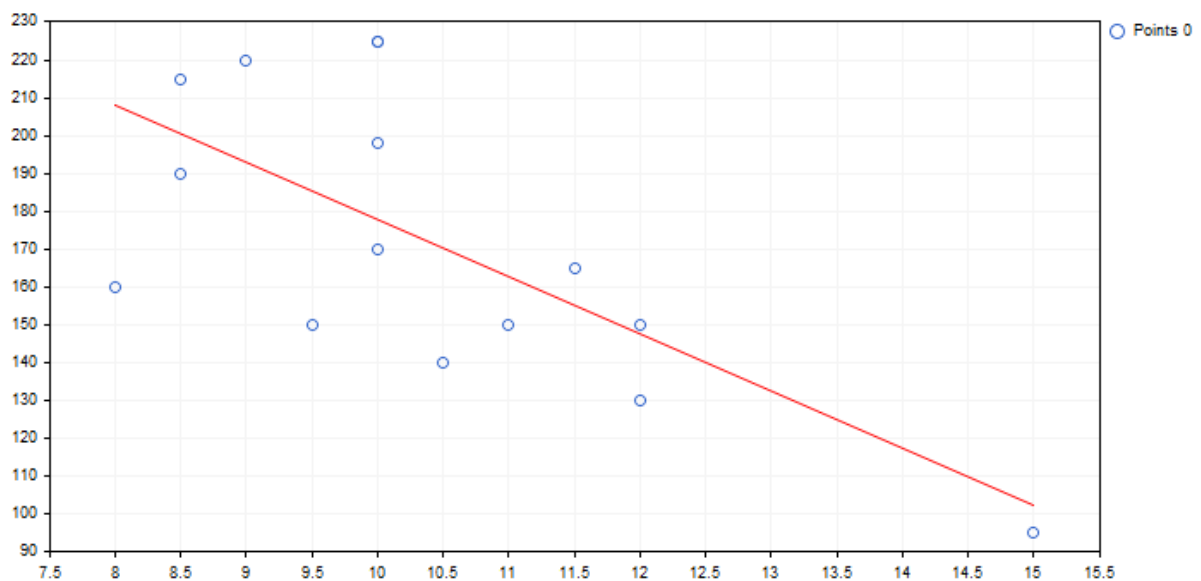
```
void TrendLineVisible(  
    const bool visible // flag value  
)
```

Parameters

visible

[in] A value of the trend line visibility flag.

Example:



Below is the code of the mentioned trend line and its plotting on the chart:


```

//+-----+
//|                                     TrendLineGraphic.mq5 |
//|               Copyright 2016, MetaQuotes Software Corp. |
//|               https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    double x[]={12.0,11.5,11.0,12.0,10.5,10.0,9.0,8.5,10.0,8.5,10.0,8.0,9.5,10.0,15.0};
    double y[]={130.0,165.0,150.0,150.0,140.0,198.0,220.0,215.0,225.0,190.0,170.0,160.0};
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"TrendLineGraphic",0,30,30,780,380))
{
    graphic.Attach(0,"TrendLineGraphic");
}
//--- create curve
CCurve *curve=graphic.CurveAdd(x,y,CURVE_POINTS);
//--- sets the curve properties
curve.TrendLineVisible(true);
curve.TrendLineColor(ColorToARGB(clrRed));
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```


Update

Update the curve coordinates.

The version for working by Y coordinate. Passed array indexes are used as X coordinates here.

```
void Update(  
    const double& y[] // Y coordinates  
)
```

This version uses X and Y coordinates.

```
void Update(  
    const double& x[], // X coordinates  
    const double& y[] // Y coordinates  
)
```

The version for working with CPoint2D points.

```
void Update(  
    const CPoint2D& points[] // Curve coordinates  
)
```

The version for working with a pointer to the CurveFunction function.

```
void Update(  
    CurveFunction function, // pointer to the function describing a curve  
    const double from,      // initial value of the function argument  
    const double to,        // final value of the function argument  
    const double step       // argument increment  
)
```

Parameters

x[]

[in] X coordinates.

y[]

[in] Y coordinates.

points[]

[in] Curve coordinates.

function

[in] A pointer to the function describing a curve

from

[in] Initial value of the function argument

to

[in] End value of the function argument

step

[in] Argument increment

Visible (Get method)

Get the flag defining if a function is visible on the chart.

```
void Visible(  
    const bool visible    //  
)
```

Return Value

A value of the flag defining a function visibility on the chart.

Visible (Set method)

Set the flag defining if a function is visible on the chart.

```
void Visible(  
    const bool visible    // flag value  
)
```

Parameters

visible

[in] A value of the flag defining a function visibility on the chart.

CGraphic

CGraphic is a base class for creating custom charts.

Description

The CGraphic class provides numerous aspects of working with custom charts.

The class stores the main chart elements, sets their parameters and performs plotting.

Also, the class stores the curves for the chart and provides various display options.

Declaration

```
class CGraphic
```

Title

```
#include <Graphics\Graphic.mqh>
```

Class methods

Method	Description
<u>Create</u>	Create a graphical resource bound to a chart object
<u>Destroy</u>	Remove a chart and destroy a graphical resource
<u>Update</u>	Display implemented changes
<u>ChartObjectName</u>	Get the name of an object bound to a chart
<u>ResourceName</u>	Get the graphical resource name
<u>XAxis</u>	Get the pointer to the X axis
<u>YAxis</u>	Get the pointer to the Y axis
<u>GapSize</u>	Get/set the size of indents between the chart elements
<u>BackgroundColor</u>	Get/set a background color
<u>BackgroundMain</u>	Get/set a chart header
<u>BackgroundMainSize</u>	Get/set a sub-header font size
<u>BackgroundMainColor</u>	Get/set a chart header color
<u>BackgroundSub</u>	Get/set a sub-header
<u>BackgroundSubSize</u>	Get/set a sub-header font size
<u>BackgroundSubColor</u>	Get/set a chart sub-header color

<u>GridLineColor</u>	Get/set a grid line color
<u>GridBackgroundColor</u>	Get/set a grid background color
<u>GridCircleRadius</u>	Get/set the dot radius in the grid nodes
<u>GridCircleColor</u>	Get/set the dot color in the grid nodes
<u>GridHasCircle</u>	Get/set the dot plotting flag in the grid nodes
<u>GridAxisLineColor</u>	Get the value of a real chart axes color.
<u>HistoryNameWidth</u>	Get/set the maximum allowed length for displaying a curve name
<u>HistoryNameSize</u>	Get/set the font size of a curve name
<u>HistorySymbolSize</u>	Get/set a size of notational convention symbols
<u>TextAdd</u>	Add a text to the chart
<u>LineAdd</u>	Add a line to the chart
<u>CurveAdd</u>	Create and add a curve to the chart
<u>CurvePlot</u>	Plot a previously created curve by index
<u>CurvePlotAll</u>	Plot all previously created curves
<u>CurveTotal</u>	Get the number of curves added to the chart
<u>CurveGetByIndex</u>	Get a curve by a specified index
<u>CurveGetByName</u>	Get a curve by a specified name
<u>CurveRemoveByIndex</u>	Remove a curve by a specified index.
<u>CurveRemoveByName</u>	Remove a curve by a specified name.
<u>CurvesTotal</u>	Get the number of curves for the given chart.
<u>MarksToAxisAdd</u>	Add a scale mark to the chart axis
<u>MajorMarkSize</u>	Get/set the size of the scale's ticks on the chart axis
<u>FontSet</u>	Set the current font parameters
<u>FontGet</u>	Get the current font parameters
<u>Attach</u>	Get/set a graphical resource and bind it to the CGraphic class instance
<u>CalculateMaxMinValues</u>	Calculate (re-calculate) minimum and maximum chart values on both axes.
<u>Height</u>	Get a chart height in pixels.
<u>IndentDown</u>	Get/set a chart indent from the lower border.
<u>IndentLeft</u>	Get/set a chart indent from the left border.

<u>IndentRight</u>	Get/set a chart indent from the right border.
<u>IndentUp</u>	Get/set a chart indent from the upper border.
<u>Redraw</u>	Redraw the chart.
<u>ResetParameters</u>	Reset the chart redrawing parameters.
<u>ScaleX</u>	Scale the value by X axis.
<u>ScaleY</u>	Scale the value by Y axis.
<u>SetDefaultParameters</u>	Set the chart parameters to default values.
<u>Width</u>	Get the chart width in pixels.

Create

Creates a graphical resource bound to a chart object.

```
bool Create(  
    const long   chart,      // chart ID  
    const string name,      // name  
    const int    subwin,    // sub-window index  
    const int    x1,        // x1 coordinate  
    const int    y1,        // y1 coordinate  
    const int    x2,        // x2 coordinate  
    const int    y2         // y1 coordinate  
)
```

Parameters

chart

[in] Chart ID.

name

[in] Name.

subwin

[in] Sub-window index.

x1

[in] X1 coordinate.

y1

[in] Y1 coordinate.

x2

[in] X2 coordinate.

y2

[in] Y2 coordinate.

Destroy

Removes a chart and destroys a graphical resource.

```
void Destroy()
```


Update

Displays implemented changes.

```
void Update(  
    const bool  redraw=true    // flag  
)
```

Parameters

redraw=true

[in] Flag value

ChartObjectName

Gets the name of an object bound to a chart.

```
string ChartObjectName()
```

Return Value

Name of an object bound to a chart.

ResourceName

Receives the name of a graphical resource.

```
string ResourceName()
```

Return Value

Name of a graphical resource.

XAxis

Returns the pointer to the X axis.

```
CAxis *XAxis ()
```

Return Value

Pointer to the X axis.

YAxis

Returns the pointer to the Y axis.

```
CAxis *YAxis ()
```

Return Value

Pointer to the Y axis.

GapSize (Get method)

Returns the size of indents between the chart elements.

```
int GapSize()
```

Return Value

Indent size in pixels.

GapSize (Set method)

Sets the size of indents between the chart elements.

```
void GapSize(  
    const int size    // indent size  
)
```

Parameters

size

[in] Indent size in pixels.

BackgroundColor (Get method)

Returns the background color.

```
color BackgroundColor()
```

BackgroundColor (Set method)

Sets the background color.

```
void BackgroundColor(  
    const color clr    // background color  
)
```

Parameters

clr

[in] Background color.

BackgroundMain (Get method)

Returns a chart header

```
string BackgroundMain()
```

BackgroundMain (Set method)

Sets a chart header text.

```
void BackgroundMain(  
    const string main    // header text  
)
```

Parameters

main

[in] Chart header text

BackgroundMainSize (Get method)

Returns the header size.

```
int BackgroundMainSize()
```

Return Value

Header font size.

BackgroundMainSize (Set method)

Sets the header size.

```
void BackgroundMainSize(  
    const int size    // header size  
)
```

Parameters

size

[in] Header font size.

BackgroundMainColor (Get method)

Returns the header color.

```
color BackgroundMainColor ()
```

Return Value

Header color.

BackgroundMainColor (Set method)

Sets the header color.

```
void BackgroundMainColor (  
    const color clr      // header color  
)
```

Parameters

clr

[in] Header color.

BackgroundSub (Get method)

Returns the sub-header.

```
string BackgroundSub ()
```

Return Value

Sub-header text.

BackgroundSub (Set method)

Sets the sub-header text.

```
void BackgroundSub (Set method) (  
    const string sub // sub-header text  
)
```

Parameters

sub

[in] Sub-header text.

BackgroundSubSize (Get method)

Returns the sub-header font size

```
int BackgroundSubSize()
```

BackgroundSubSize (Get method)

Sets the sub-header size

```
void BackgroundSubSize(  
    const int size // sub-header font size  
)
```

Parameters

size

[in] Sub-header font size

BackgroundSubColor (Get method)

Returns the sub-header color.

```
color BackgroundSubColor()
```

BackgroundSubColor (Set method)

Sets the sub-header color.

```
void BackgroundSubColor(  
    const color clr    // sub-header color  
)
```

Parameters

clr

[in] Sub-header color.

GridLineColor (Get method)

Returns the grid line color

```
color GridLineColor()
```

Return Value

Grid line color.

GridLineColor (Set method)

Sets the grid line color.

```
void GridLineColor(  
    const color clr    // line color  
)
```

Parameters

clr

[in] Grid line color.

GridBackgroundColor (Get method)

Returns the grid background color

```
color GridBackgroundColor ()
```

Return Value

Grid background color

GridBackgroundColor (Set method)

Sets the grid background color

```
void GridBackgroundColor (  
    const color clr      // grid background color  
)
```

Parameters

clr

[in] Grid background color

GridCircleRadius (Get method)

Returns radius of dots in the grid nodes.

```
int GridCircleRadius ()
```

Return Value

Dot radius in pixels.

GridCircleRadius (Set method)

Sets the dot radius in the grid nodes

```
void GridCircleRadius (  
    const int r      // radius  
)
```

Parameters

r

[in] Dot radius in pixels.

GridCircleColor (Get method)

Returns color of dots in the grid nodes.

```
color GridCircleColor()
```

Return Value

Dot color.

GridCircleColor (Set method)

Sets the dot color in the grid nodes.

```
void GridCircleColor(  
    const color clr    // dot color  
)
```

Parameters

clr

[in] Dot color.

GridHasCircle (Get method)

Returns the flag defining whether the dots in the grid nodes should be displayed.

```
bool GridHasCircle()
```

Return Value

The flag value.

Note

true — display the dots

false — do not display the dots

GridHasCircle (Set method)

Sets the flag defining whether the dots in the grid nodes should be displayed.

```
void GridHasCircle(  
    const bool has  
)
```

Parameters

has

[in] Flag value.

Note

true — display the dots

false — do not display the dots

GridAxisLineColor (Get method)

Get the value of a real chart axes color.

```
uint GridAxisLineColor()
```

Return Value

Color of real chart axes.

GridAxisLineColor (Set method)

Set the value of a real chart axes color.

```
void GridAxisLineColor(  
    const uint clr // chart axes color  
)
```

Parameters

clr

[in] Color of real chart axes.

HistoryNameWidth (Get method)

Returns the maximum allowed length for displaying a curve name.

```
int HistoryNameWidth()
```

Return Value

Maximum length in pixels.

Note

If the curve name exceeds the maximum allowed length, it is truncated and dots are added to its end.

HistoryNameWidth (Set method)

Sets the maximum allowed length for displaying a curve name.

```
void HistoryNameWidth(  
    const int width // maximum length  
)
```

Parameters

width

[in] Maximum length in pixels.

Note

If the curve name exceeds the maximum allowed length, it is truncated and dots are added to its end.

HistoryNameSize (Get method)

Returns the font size of the curve name.

```
int HistoryNameSize()
```

Return Value

Font size of the curve name.

HistoryNameSize (Set method)

Sets the font size of the curve name.

```
void HistoryNameSize (Set method) (  
    const int size // name font size  
)
```

Parameters

size

[in] Name font size.

HistorySymbolSize (Get method)

Returns a size of a chart's notational convention symbols

```
int HistorySymbolSize()
```

Return Value

Size of notational convention symbols

HistorySymbolSize (Set method)

Sets a size of a chart's notational convention symbols

```
void HistorySymbolSize(  
    const int size // symbol size  
)
```

Parameters

size

[in] Size of notational convention symbols.

TextAdd

Adds a text to a chart.

Version for working with X and Y coordinates

```
void TextAdd(  
    const int      x,           // X coordinate  
    const int      y,           // Y coordinate  
    const string   text,        // text  
    const uint     clr,         // color  
    const uint     alignment=0  // alignment  
)
```

Version for CPoint

```
void TextAdd(  
    const CPoint   &point,      // point coordinate  
    const string   text,        // text  
    const uint     clr,         // color  
    const uint     alignment=0  // alignment  
)
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

&point

[in] Point coordinate.

text

[in] Text.

clr

[in] Color.

alignment=0

[in] Alignment.

LineAdd

Adds a line to a chart.

This version uses X and Y coordinates

```
void LineAdd(  
    const int   x1,           // x1 coordinate  
    const int   y1,           // y1 coordinate  
    const int   x2,           // x2 coordinate  
    const int   y2,           // y2 coordinate  
    const uint  clr,          // color  
    const uint  style         // style  
)
```

Version for CPoint

```
void LineAdd2(  
    const CPoint &point1,     // first point coordinate  
    const CPoint &point2,     // second point coordinate  
    const uint  clr,          // color  
    const uint  style         // style  
)
```

Parameters

x1

[in] X1 coordinate.

y1

[in] Y1 coordinate.

x2

[in] X2 coordinate.

y2

[in] Y2 coordinate.

&point1

[in] First point coordinate.

&point2

[in] Second point coordinate.

clr

[in] Color.

style

[in] Style.

CurveAdd

Create and add a new curve to the chart.

This version uses the Y coordinate (a curve color is set automatically)

```
CCurve* CurveAdd(
    const double    &y[],           // Y coordinates
    ENUM_CURVE_TYPE type,           // curve type
    const string    name=NULL       // curve name
)
```

Note

Y array indices are used as X coordinates for the curve.

This version uses the X and Y coordinates (a curve color is set automatically)

```
CCurve* CurveAdd(
    const double    &x[],           // X coordinates
    const double    &y[],           // Y coordinates
    ENUM_CURVE_TYPE type,           // curve type
    const string    name=NULL       // curve name
)
```

The version for working with CPoint2D dots (curve color is set automatically)

```
CCurve* CurveAdd(
    const CPoint2D  &points[],       // dot coordinates
    ENUM_CURVE_TYPE type,           // curve type
    const string    name=NULL       // curve name
)
```

Version for working with the pointer to the CurveFunction function (curve color is set automatically)

```
CCurve* CurveAdd(
    CurveFunction    function,       // pointer to the function
    const double     from,           // initial value of the argument
    const double     to,             // final value of the argument
    const double     step,           // increment by the argument
    ENUM_CURVE_TYPE type,           // curve type
    const string     name=NULL       // curve name
)
```


Version for working by Y coordinate (a curve color is set by a user)

```
CCurve* CurveAdd(  
    const double    &y[],           // Y coordinates  
    const uint      clr,           // curve color  
    ENUM_CURVE_TYPE type,         // curve type  
    const string    name=NULL      // curve name  
)
```

Note

Y array indices are used as X coordinates for the curve.

This version uses the X and Y coordinates (a curve color is set by a user)

```
CCurve* CurveAdd(  
    const double    &x[],           // X coordinates  
    const double    &y[],           // Y coordinates  
    const uint      clr,           // curve color  
    ENUM_CURVE_TYPE type,         // curve type  
    const string    name=NULL      // curve name  
)
```

The version for working with CPoint2D dots (curve color is set by a user)

```
CCurve* CurveAdd(  
    const CPoint2D  &points[],      // dot coordinates  
    const uint      clr,           // curve color  
    ENUM_CURVE_TYPE type,         // curve type  
    const string    name=NULL      // curve name  
)
```

Version for working with the pointer to the CurveFunction function (curve color is set by a user)

```
CCurve* CurveAdd(  
    CurveFunction    function,      // pointer to the function  
    const double     from,          // initial value of the argument  
    const double     to,           // final value of the argument  
    const double     step,         // increment by the argument  
    const uint      clr,           // curve color  
    ENUM_CURVE_TYPE type,         // curve type  
    const string    name=NULL      // curve name  
)
```


Parameters*&x[]*

[in] X coordinate.

&y[]

[in] Y coordinate.

&points[]

[in] Coordinates of dots.

function

[in] Pointer to the function.

from

[in] Initial value of the argument.

to

[in] Final value of the argument.

step

[in] Increment by the argument.

type

[in] Curve type.

name=NULL

[in] Curve name.

clr

[in] Curve color.

Return Value

Pointer to the created curve.

CurvePlot

Displays the previously created curve with a specified index.

```
bool CurvePlot(  
    const int index    // index  
)
```

Parameters

index

[in] Curve index

Return Value

true - successful, otherwise - false.

CurvePlotAll

Displays all curves previously added to a chart.

```
bool CurvePlotAll()
```

Return Value

true - successful, otherwise - false.

CurveTotal

Returns the number of curves added to the chart.

```
int CurveTotal()
```

Return Value

Number of curves.

CurveGetByIndex

Gets the curve by a specified index.

```
CCurve* CurveGetByIndex(  
    const int index    // curve index  
)
```

Parameters

index

[in] Curve index.

Return Value

Pointer to the curve with a specified index.

CurveGetByName

Gets a curve by a specified name.

```
CCurve* CurveGetByName(  
    const string name    // curve name  
)
```

Parameters

name

[in] Curve name.

Return Value

Pointer to the first found curve having a specified name.

CurveRemoveByIndex

Remove a curve by a specified index.

```
bool CurveRemoveByIndex(  
    const int index    // curve index  
)
```

Parameters

index

[in] Index of the curve to be removed.

Return Value

true — successful, otherwise — false.

CurveRemoveByName

Remove a curve by a specified name.

```
bool CurveRemoveByName(  
    const string name    // curve name  
)
```

Parameters

name

[in] Name of the curve to be removed.

Return Value

true — successful, otherwise — false.

CurvesTotal

Get the number of curves for the given chart.

```
int CurvesTotal()
```

Return Value

Number of curves.

Note

All curves on the current chart are considered regardless of drawing and visibility style.

MarksToAxisAdd

Add a scale mark (ticks) to the specified chart axis.

```
bool MarksToAxisAdd(  
    const double      &marks[],           // tick coordinates  
    const int         mark_size,          // tick size  
    ENUM_MARK_POSITION position,          // tick location  
    const int         dimension=0         // dimension  
)
```

Parameters

&marks[]

[in] Tick coordinates

mark_size

[in] Tick size

position

[in] Tick location

dimension=0

[in] 0 – adding to X axis,

1 – adding to Y axis

Return Value

true - successful, otherwise - false.

MajorMarkSize (Get method)

Returns the size of the scale's ticks on the coordinate axes.

```
int MajorMarkSize()
```

MajorMarkSize (Set method)

Returns the size of the scale's ticks on the coordinate axes.

```
void MajorMarkSize(  
    const int size // tick size  
)
```

Parameters

size

[in] Tick size in pixels.

FontSet

Sets the current font parameters.

```
bool FontSet(  
    const string name,      // name  
    const int size,         // size  
    const uint flags=0,     // flags  
    const uint angle=0      // angle  
)
```

Parameters

name

[in] Name.

size

[in] Size.

flags=0

[in] Flags.

angle=0

[in] Angle.

Return Value

true - successful, otherwise - false.

FontGet

Gets the current font parameters.

```
void FontGet(  
    string  &name,      // name  
    int     &size,      // size  
    uint    &flags,     // flags  
    uint    &angle      // angle  
)
```

Parameters

&name

[out] Name.

&size

[out] Size.

&flags

[out] Flags.

&angle

[out] Angle.

Attach

The version for getting a graphical resource from the OBJ_BITMAP_LABEL object and binding it to the CGraphic class instance:

```
bool Attach(  
    const long   chart_id,      // chart ID  
    const string objname       // graphical object name  
)
```

The version for creating a graphical resource for the OBJ_BITMAP_LABEL object and binding it to the CGraphic class instance:

```
bool Attach(  
    const long   chart_id,      // chart ID  
    const string objname,       // graphical object name  
    const int    width,         // image width  
    const int    height        // image height  
)
```

Parameters

chart_id

[in] Chart ID.

objname

[in] Name of the graphical object.

width

[in] Image width in the resource.

height

[in] Image height in the resource.

Return Value

true — successful, false — failed to bind the object.

CalculateMaxMinValues

Calculate (re-calculate) minimum and maximum chart values on both axes.

```
void CalculateMaxMinValues ()
```


Height

Get a chart height in pixels.

```
int Height()
```

Return Value

Chart height in pixels.

IndentDown (Get method)

Get a chart indent from the lower border.

```
int IndentDown()
```

Return Value

Indent size in pixels.

IndentDown (Set method)

Set a chart indent from the lower border.

```
void IndentDown(  
    const int down // indent size  
)
```

Parameters

down

[in] Indent size in pixels.

IndentLeft (Get method)

Get a chart indent from the left border.

```
int IndentLeft()
```

Return Value

Indent size in pixels.

IndentLeft (Set method)

Set a chart indent from the left border.

```
void IndentLeft(  
    const int left    // indent size  
)
```

Parameters

left

[in] Indent size in pixels.

IndentRight (Get method)

Get a chart indent from the right border.

```
int IndentRight()
```

Return Value

Indent size in pixels.

IndentRight (Set method)

Set a chart indent from the right border.

```
void IndentRight(  
    const int right // indent size  
)
```

Parameters

right

[in] Indent size in pixels.

IndentUp (Get method)

Get a chart indent from the upper border.

```
int IndentUp()
```

Return Value

Indent size in pixels.

IndentUp (Set method)

Set a chart indent from the upper border.

```
void IndentUp(  
    const int up    // indent size  
)
```

Parameters

up

[in] Indent value in pixels.

Redraw

Redraw the chart.

```
bool Redraw(  
    const bool rescale=false    // flag value  
)
```

Parameters

rescale=false

[in] The flag indicating if a chart should be re-scaled.

Return Value

true - successful, otherwise - false.

ResetParameters

Reset the chart redrawing parameters.

```
void ResetParameters ()
```


ScaleX

Scale the value by X axis.

```
virtual int ScaleX(  
    double x      // value by X axis  
)
```

Parameters

x

[in] Real value by X axis.

Return Value

A value in pixels.

Note

Scale a real value to pixels for displaying on the chart.

ScaleY

Scale the value by Y axis.

```
virtual int ScaleY(  
    double y      // value by Y axis  
)
```

Parameters

y

[in] Real value by Y axis.

Return Value

A value in pixels.

Note

Scale a real value to pixels for displaying on the chart.

SetDefaultParameters

Set the chart parameters to default values.

```
void SetDefaultParameters ()
```


Width

Get the chart width in pixels.

```
int Width()
```

Return Value

Chart width in pixels.

Technical Indicators and Timeseries

This section contains the technical details of the technical indicator and timeseries classes and description of the corresponding components of the standard MQL5 library.

The use of the technical indicator and timeseries classes will save time in developing applications (scripts, Expert Advisors).

The MQL5 Standard Library (in terms of technical indicators and timeseries) is located in the working directory of the terminal in the Include\Indicators folder.

Class/group	Description
Base classes	Group of base and auxiliary classes
Timeseries classes	Group of timeseries classes
Trend Indicators	Group of Trend indicator classes
Oscillators	Group of Oscillator indicator classes
Volume Indicators	Group of Volume indicator classes
Bill Williams Indicators	Group of Bill Williams indicator classes
Custom indicators	Custom indicator class

Base and Auxiliary Technical Indicator and Timeseries Classes

This section contains the technical details of base and auxiliary technical indicator and timeseries classes and description of the corresponding components of the Standard MQL5 library.

Class/group	Description
<u>CSpreadBuffer</u>	Historical spread buffer class
<u>CTimeBuffer</u>	Historical opening prices buffer class
<u>CTickVolumeBuffer</u>	Historical tick volumes buffer class
<u>CRealVolumeBuffer</u>	Historical real volumes buffer class
<u>CDoubleBuffer</u>	Base class of double type data buffer
<u>COpenBuffer</u>	Opening bar prices buffer class
<u>CHighBuffer</u>	High bar prices buffer class
<u>CLowBuffer</u>	Low bar prices buffer class
<u>CCloseBuffer</u>	Closing bar prices buffer class
<u>CIndicatorBuffer</u>	Technical indicator buffer class
<u>CSeries</u>	Base class for access to timeseries data
<u>CPriceSeries</u>	Base class for access to price data
<u>CIndicator</u>	Base class of technical indicator
<u>CLindicators</u>	Technical indicator and timeseries collection

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

CSpreadBuffer

CSpreadBuffer is a class for simplified access to spreads of the bars in the history.

Description

The CSpreadBuffer class provides a simplified access to spreads of the bars in the history.

Declaration

```
class CSpreadBuffer: public CArrayInt
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayInt](#)

CSpreadBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayInt

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, Minimum, Maximum, [Update](#), [Shift](#), [Delete](#),

[DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#),
[SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Size

Sets buffer size.

```
void Size(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void SetSymbolPeriod(  
    const string      symbol,      // symbol  
    const ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] New symbol.

period

[in] New period ([ENUM_TIMEFRAMES](#) enumeration).

At

Gets the buffer element.

```
int At(  
    const int index    // index  
    ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CTimeBuffer

CTimeBuffer is a class for simplified access to opening times of the bars in the history.

Description

The CTimeBuffer class provides a simplified access to opening times of the bars in the history.

Declaration

```
class CTimeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayLong](#)

CTimeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayLong

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, Minimum, Maximum, [Update](#), [Shift](#), [Delete](#),

[DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#),
[SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Size

Sets buffer size.

```
void Size(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void SetSymbolPeriod(  
    const string      symbol,      // symbol  
    const ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] New symbol.

period

[in] New period ([ENUM_TIMEFRAMES](#) enumeration).

At

Gets the buffer element.

```
long At(  
    const int index    // index  
) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CTickVolumeBuffer

CTickVolumeBuffer is a class for simplified access to tick volumes of bars in the history.

Description

The CTickVolumeBuffer class provides a simplified access to tick volumes of bars in the history.

Declaration

```
class CTickVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayLong](#)

CTickVolumeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayLong

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, Minimum, Maximum, [Update](#), [Shift](#), [Delete](#),

[DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#),
[SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Size

Sets buffer size.

```
void Size(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void SetSymbolPeriod(  
    const string      symbol,      // symbol  
    const ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] New symbol.

period

[in] New period ([ENUM_TIMEFRAMES](#) enumeration).

At

Gets the buffer element.

```
long At(  
    const int index    // index  
    ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CRealVolumeBuffer

CRealVolumeBuffer is a class for simplified access to real volumes of bars in the history.

Description

CTickVolumeBuffer class provides a simplified access to real volumes of bars in the history.

Declaration

```
class CRealVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayLong](#)

CRealVolumeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayLong

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, Minimum, Maximum, [Update](#), [Shift](#), [Delete](#),

[DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#),
[SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Size

Sets buffer size.

```
void Size(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void SetSymbolPeriod(  
    const string      symbol,      // symbol  
    const ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] New symbol.

period

[in] New period ([ENUM_TIMEFRAMES](#) enumeration).

At

Gets the buffer element.

```
long At(  
    const int index    // index  
    ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CDoubleBuffer

CDoubleBuffer is a base class for simplified access to data buffers of double type.

Description

The CDoubleBuffer class provides a simplified access to the data of the buffer of double type.

Declaration

```
class CDoubleBuffer: public CArrayDouble
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayDouble](#)

CDoubleBuffer

Direct descendants

[CCloseBuffer](#), [CHighBuffer](#), [CIndicatorBuffer](#), [CLowBuffer](#), [COpenBuffer](#)

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Size

Sets buffer size.

```
void Size(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void SetSymbolPeriod(  
    const string      symbol,      // symbol  
    const ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] New symbol.

period

[in] New period ([ENUM_TIMEFRAMES](#) enumeration).

At

Gets the buffer element.

```
double At(  
    const int index    // index  
) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

COpenBuffer

COpenBuffer is a class for simplified access to open prices of bars in the history.

Description

COpenBuffer class provides a simplified access to open prices of bars in the history.

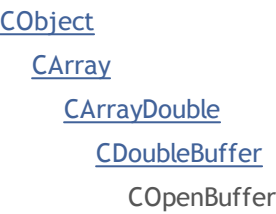
Declaration

```
class COpenBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Methods inherited from class CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CHighBuffer

CHighBuffer is a class for simplified access to high prices of bars in the history.

Description

CHighBuffer class provides a simplified access to high prices of bars in the history.

Declaration

```
class CHighBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayDouble](#)

[CDoubleBuffer](#)

CHighBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Methods inherited from class CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CLowBuffer

CLowBuffer is a class for simplified access to low prices of bars in the history.

Description

The CLowBuffer class provides a simplified access to low prices of bars in the history.

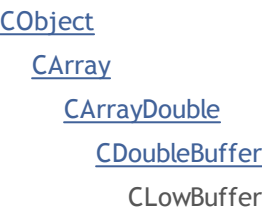
Declaration

```
class CLowBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Methods inherited from class CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CCloseBuffer

CCloseBuffer is a class for simplified access to close prices of bars in the history.

Description

CCloseBuffer class provides a simplified access to close prices of bars in the history.

Declaration

```
class CCloseBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayDouble](#)

[CDoubleBuffer](#)

CCloseBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Methods inherited from class CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

Refresh

Updates the buffer.

```
virtual bool Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CIndicatorBuffer

CIndicatorBuffer is a class for simplified access to the data of the indicator's buffer.

Description

CIndicatorBuffer class provides the simplified access to the data buffer of technical indicator.

Declaration

```
class CIndicatorBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\Indicator.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayDouble](#)

[CDoubleBuffer](#)

CIndicatorBuffer

Class Methods by Groups

Attributes	
Offset	Gets/sets offset of the buffer
Name	Gets/sets buffer name
Data Access	
At	Gets buffer's element
Data Update	
Refresh	Updates the buffer
RefreshCurrent	Updates only the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#),

[Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Methods inherited from class CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

Offset

Gets offset of the buffer.

```
int Offset() const
```

Return Value

Buffer offset.

Offset

Sets offset of the buffer.

```
void Offset(  
    const int offset    // offset  
)
```

Parameters

offset

[in] New buffer offset.

Name

Gets the name of the buffer.

```
string Name() const
```

Return Value

Name of the buffer.

Name

Sets the name of the buffer.

```
void Name(  
    const string name    // name  
)
```

Parameters

name

[in] New name of the buffer.

At

Gets buffer element.

```
double At(  
    int index    // index  
) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the whole buffer.

```
bool Refresh(  
    const int handle,    // indicator handle  
    const int num        // buffer number  
)
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Buffer index of the indicator.

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) buffer element.

```
bool RefreshCurrent(  
    const int handle,      // handle of the indicator  
    const int num          // buffer number  
)
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Indicator buffer number.

Return Value

true - successful, false - cannot update the buffer.

CSeries

CSeries is a base class for an access to the timeseries data of the Standard Library.

Description

CSeries class provides the simplified access to all the MQL5 API general functions related to working with the series data for all its descendants (timeseries and indicator classes).

Declaration

```
class CSeries: public CArrayObj
```

Title

```
#include <Indicators\Series.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

CSeries

Direct descendants

[CIndicator](#), [CiRealVolume](#), [CiSpread](#), [CiTickVolume](#), [CiTime](#), [CPriceSeries](#)

Class Methods by Groups

Attributes	
Name	Gets the name of timeseries or indicator
BuffersTotal	Gets the number of buffers of timeseries or indicator
Timeframe	Gets the timeframe flag of timeseries or indicator
Symbol	Gets the symbol of timeseries or indicator
Period	Gets the period of timeseries or indicator
RefreshCurrent	Sets/resets the flag of updating the current data
Data Access	
virtual BufferResize	Sets buffer size of timeseries or indicator
Data Update	
virtual Refresh	Update the data of timeseries or indicator

[PeriodDescription](#)Transforms [ENUM_TIMEFRAMES](#) into a string**Methods inherited from class CObject**Prev, Prev, Next, Next, [Compare](#)**Methods inherited from class CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Methods inherited from class CArrayObj**[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Name

Gets the name of timeseries or indicator.

```
string Name() const
```

Return Value

The name of timeseries or indicator.

BuffersTotal

Gets the number of buffers of timeseries or indicator.

```
int BuffersTotal() const
```

Return Value

The number of buffers of timeseries or indicator.

Note

The timeseries has only one buffer.

Timeframe

Gets the timeframe flag of timeseries or indicator.

```
int Timeframe() const
```

Return Value

The timeframe flag of timeseries or indicator.

Note

It is generated as the object visibility flags.

Symbol

Gets the symbol of timeseries or indicator.

```
string Symbol() const
```

Return Value

The symbol of timeseries or indicator.

Period

Gets the period of timeseries or indicator.

```
ENUM_TIMEFRAMES Period() const
```

Return Value

The period (value of [ENUM_TIMEFRAMES](#) enumeration) of timeseries or indicator.

RefreshCurrent

Sets a flag to constantly update the current values of timeseries or indicator.

```
string RefreshCurrent(  
    const bool flag    // value  
)
```

Parameters

flag

[in] New flag.

Return Value

None.

BufferSize

Returns the amount of data available in timeseries buffer or indicator buffer.

```
int BufferSize() const
```

Return Value

Amount of data available in timeseries buffer or indicator buffer.

BufferResize

Sets buffer size of timeseries or indicator.

```
virtual bool BufferResize(  
    const int size    // size  
)
```

Parameters

size

[in] New size of the buffers.

Return Value

true - successful, otherwise - false.

Note

All the timeseries or indicator buffers have the same size.

Refresh

Updates the data of timeseries or indicator.

```
virtual void Refresh(  
    const int flags    // flags  
)
```

Parameters

flags

[in] Timeframes to update (flag).

PeriodDescription

Gets the string representation of the specified [ENUM_TIMEFRAMES](#) enumeration.

```
string PeriodDescription(  
    const int val=0      // value  
)
```

Parameters

val=0

[in] Value to convert.

Return Value

The string representation of the specified [ENUM_TIMEFRAMES](#) enumeration.

Note

If the value is not specified or equal to zero, the timeframe of timeseries or indicator is transformed into a string.

CPriceSeries

CPriceSeries is a base class for access to the price data.

Description

CSeries class provides the simplified access to MQL5 API general functions for working with price data to all its descendants.

Declaration

```
class CPriceSeries: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CPriceSeries

Direct descendants

[CiClose](#), [CiHigh](#), [CiLow](#), [CiOpen](#)

Class Methods by Groups

Create	
virtual BufferResize	Sets size of the series buffer
Data Access	
virtual GetData	Gets the specified series buffer element
Data Update	
virtual Refresh	Updates timeseries data
Search for Extreme Values	
virtual MinIndex	Gets the index of minimal value in the specified range
virtual MinValue	Gets the minimal value in the specified range
virtual MaxIndex	Gets the index of maximal value in the specified range
virtual MaxValue	Gets the maximal value in the specified range

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

BufferResize

Sets the series buffer size.

```
virtual void BufferResize(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

Return Value

true - successful, otherwise - false.

GetData

Gets the specified series buffer element.

```
double GetData(  
    const int index    // index  
) const
```

Parameters

index

[in] Index of a buffer element.

Return Value

The series buffer element, or [EMPTY_VALUE](#).

Refresh

Updates the timeseries data

```
virtual void Refresh(  
    const int flags=OBJ_ALL_PERIODS    // flags  
)
```

Parameters

flags=OBJ_ALL_PERIODS

[in] Timeframe flags.

MinIndex

Gets the index of minimal value in the specified range.

```
virtual int MinIndex(  
    const int start,      // size  
    const int count       // number  
) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

The index of minimal value of a series buffer in the specified range, or -1.

MinValue

Gets the minimal value in the specified range.

```
virtual double MinValue(  
    const int    start,      // size  
    const int    count,      // number  
    int&         index       // reference  
) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for placing the found element's index value.

Return Value

The minimal value of the series buffer in the specified range, or [EMPTY_VALUE](#).

Note

The index of the found element is stored by index reference.

MaxIndex

Gets the index of maximal value in the specified range.

```
virtual int MaxIndex(  
    const int start,    // index  
    const int count     // number  
) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

The index of the maximal value of the series buffer in the specified range, or -1.

MaxValue

Gets the maximal value in the specified range.

```
virtual double MaxValue(  
    const int    start,      // size  
    const int    count,     // amount  
    int&         index       // reference  
) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for placing the found element's index value.

Return Value

The maximal value of a series buffer in the specified range, or [EMPTY_VALUE](#).

Note

The index of the found element is stored by index reference.

CIndicator

CIndicator is a base class for technical indicator classes of the MQL5 standard library.

Description

The CIndicator class provides the simplified access for all of its descendants to general MQL5 API technical indicator functions.

Declaration

```
class CIndicator: public CSeries
```

Title

```
#include <Indicators\Indicator.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CIndicator

Direct descendants

[CiAC](#), [CiAD](#), [CiADX](#), [CiADXWilder](#), [CiAlligator](#), [CiAMA](#), [CiAO](#), [CiATR](#), [CiBands](#), [CiBearsPower](#), [CiBullsPower](#), [CiBWMFI](#), [CiCCI](#), [CiChaikin](#), [CiCustom](#), [CiDEMA](#), [CiDeMarker](#), [CiEnvelopes](#), [CiForce](#), [CiFractals](#), [CiFrAMA](#), [CiGator](#), [CiIchimoku](#), [CiMA](#), [CiMACD](#), [CiMFI](#), [CiMomentum](#), [CiOBV](#), [CiOsMA](#), [CiRSI](#), [CiRVI](#), [CiSAR](#), [CiStdDev](#), [CiStochastic](#), [CiTEMA](#), [CiTriX](#), [CiVIDyA](#), [CiVolumes](#), [CiWPR](#)

Class Methods by Groups

Attributes	
Handle	Gets the indicator's handle.
Status	Gets the status of the indicator.
FullRelease	Sets a flag to release the handle.
Creation	
Create	Creates the indicator
BufferResize	Sets the new buffer sizes
Data Access	
GetData	Copies the data from the indicator buffer
Data Update Methods	
Refresh	Updates the indicator data

Finding Min/Max Values	
Minimum	Gets the index of minimal value in a specified range.
MinValue	Gets the minimal value in a specified range.
Maximum	Gets the index of maximal value in a specified range.
MaxValue	Gets the maximal value in a specified range.
Conversion of Enumerations	
MethodDescription	Converts ENUM_MA_METHOD into a string
PriceDescription	Converts ENUM_APPLIED_PRICE into a string
VolumeDescription	Converts ENUM_APPLIED_VOLUME into a string
Working with chart	
AddToChart	Adds an indicator to the chart
DeleteFromChart	Deletes an indicator from the chart

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Handle

Gets the indicator's handle.

```
int Handle() const
```

Return Value

Handle of the indicator.

Status

Gets the status of the indicator.

```
string Status() const
```

Return Value

The status of indicator creation.

FullRelease

Sets a flag to release the handle.

```
void FullRelease(  
    const bool flag=true    // flag  
)
```

Parameters

flag

[in] New value of the handle release flag.

Create

Creates the indicator of the specified type with the specified parameters.

```
bool Create(  
    const string      symbol,          // symbol  
    const ENUM_TIMEFRAMES period,      // period  
    const ENUM_INDICATOR type,         // type  
    const int         num_params,      // number of parameters  
    const MqlParam&   params[]        // array of parameters  
)
```

Parameters

symbol

[in] Indicator symbol.

period

[in] Indicator timeframe ([ENUM_TIMEFRAMES](#) enumeration).

type

[in] Indicator type ([ENUM_INDICATOR](#) enumeration).

num_params

[in] Number of indicator's parameters.

params

[in] Reference to the parameters array for the indicator.

Return Value

true - successful, false - cannot create the indicator.

BufferResize

Sets the sizes of the indicator buffers.

```
virtual bool BufferResize(  
    const int size    // size  
)
```

Parameters

size

[in] New buffer size.

Return Value

true - successful, otherwise - false.

Note

All the indicator buffers have the same size.

BarsCalculated

Returns the amount of calculated data for the indicator.

```
int BarsCalculated() const;
```

Return Value

Returns the amount of calculated data in the indicator buffer, or -1 in the case of error (data is not calculated yet).

GetData

Gets the specified element of the specified buffer of the indicator. [Refresh\(\)](#) should be called for working with recent data before using the method.

```
double  GetData (
    const int  buffer_num,      // buffer number
    const int  index           // index
) const
```

Parameters

buffer_num

[in] Indicator buffer number.

index

[in] Indicator buffer element index.

Return Value

value - success, [EMPTY_VALUE](#) - cannot receive the data.

GetData

Gets the data from the indicator's buffer by starting position and number.

```
int  GetData (
    const int      start_pos,      // position
    const int      count,          // number
    const int      buffer_num,     // buffer number
    double&        buffer[]        // array
) const
```

Parameters

start_pos

[in] Starting position of the indicator buffer.

count

[in] Number of indicator buffer elements.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified indicator buffer - success, otherwise -1.

GetData

Gets the data from the indicator buffer by start time and number.


```
int  GetData(
    const datetime  start_time,    // starting time
    const int       count,         // amount
    const int       buffer_num,    // buffer number
    double&         buffer[]       // array
) const
```

Parameters

start_time

[in] Indicator buffer element starting time.

count

[in] Number of indicator buffer elements.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified buffer, otherwise -1.

GetData

Gets the data from the indicator buffer by start and end time.

```
int  GetData(
    const datetime  start_time,    // start time
    const datetime  stop_time,     // end time
    const int       buffer_num,    // number of buffer
    double&         buffer[]       // array
) const
```

Parameters

start_time

[in] Indicator buffer initial element time.

stop_time

[in] Indicator buffer end element time.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified buffer - success, otherwise -1.

Refresh

Updates the indicator data. It is recommended calling the method before using [GetData\(\)](#).

```
virtual void Refresh(  
    int flags=OBJ_ALL_PERIODS    // flags  
)
```

Parameters

flags=OBJ_ALL_PERIODS

[in] Timeframe update flags.

Minimum

Returns the index of minimal element of the specified buffer in a specified range.

```
int Minimum(  
    const int  buffer_num,      // buffer number  
    const int  start,          // starting index  
    const int  count           // number  
) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

Index of the minimal element of the specified buffer in a specified range.

MinValue

Returns the value of minimal element of the specified buffer in a specified range.

```
double MinValue(  
    const int    buffer_num,      // buffer number  
    const int    start,           // starting index  
    const int    count,           // number  
    int&         index            // reference  
) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for storing the found element index value.

Return Value

The value of the minimal element of the specified buffer in a specified range.

Note

The index of the found element is stored by index reference.

Maximum

Returns the index of maximal element of the specified buffer in a specified range.

```
int Maximum(  
    const int  buffer_num,      // buffer number  
    const int  start,          // starting index  
    const int  count           // number  
) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

Index of the maximal element of the specified buffer in a specified range.

MaxValue

Returns the value of maximal element of the specified buffer in a specified range.

```
double MaxValue(  
    const int    buffer_num,      // buffer number  
    const int    start,           // starting index  
    const int    count,           // number  
    int&         index            // reference  
) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for storing the found element index value.

Return Value

The value of the maximal element of the specified buffer in a specified range.

Note

The index of maximal buffer element is stored by index reference.

MethodDescription

Converts [ENUM_MA_METHOD](#) enumeration value to a string.

```
string MethodDescription(  
    const int val    // value  
) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to [ENUM_MA_METHOD](#) enumeration value.

PriceDescription

Converts [ENUM_APPLIED_PRICE](#) enumeration value to a string.

```
string PriceDescription(  
    const int val    // value  
) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to [ENUM_APPLIED_PRICE](#) enumeration value.

VolumeDescription

Converts [ENUM_APPLIED_VOLUME](#) enumeration value to a string.

```
string VolumeDescription(  
    const int val    // value  
) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to [ENUM_APPLIED_VOLUME](#) enumeration value.

AddToChart

Adds the indicator to the chart.

```
bool AddToChart(  
    const long chart,      // chart ID  
    const int subwin       // subwindow index  
)
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow index.

Return Value

true - successful, false - cannot add the indicator to the chart.

DeleteFromChart

Deletes the indicator from the chart.

```
bool DeleteFromChart(  
    const long chart,      // chart ID  
    const int subwin       // subwindow index  
)
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow index.

Return Value

true - successful, false - cannot remove the indicator from the chart.

Indicators

The Indicators is a class for collecting instances of timeseries and technical indicators classes.

Description

Indicators class provides creation of the technical indicators class instances, their storage and management (data synchronization, handle and memory management).

Declaration

```
class CIndicators: public CArrayObj
```

Title

```
#include <Indicators\Indicators.mqh>
```

Class Methods by Groups

Create	
Create	Creates an indicator
Data Update	
Refresh	Updates indicator data

Create

Creates the indicator of the specified type with the specified parameters.

```
CIndicator* Create(  
    const string      symbol,      // symbol name  
    const ENUM_TIMEFRAMES period,    // period  
    const ENUM_INDICATOR type,      // type  
    const int         count,      // number of parameters  
    const MqlParam&   params      // parameters array  
)
```

Parameters

symbol

[in] Indicator symbol name.

period

[in] Indicator timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

type

[in] Indicator type ([ENUM_INDICATOR](#) enumeration value).

count

[in] Number of parameters for the indicator.

params

[in] Reference to the parameters array for the indicator.

Return Value

Reference to the created indicator - successful, [NULL](#) - cannot create the indicator.

Refresh

Updates data for all timeseries and technical indicators in the collection.

```
int Refresh()
```

Return Value

Updated timeframe flags (formed as object visibility flags).

Timeseries classes

This group of chapters contains technical details of timeseries classes of the MQL5 Standard Library and descriptions of all its key components.

Class	Description
<u>CiSpread</u>	Provides an access to spread historical data
<u>CiTime</u>	Provides an access to open times of the bars in the history
<u>CiTickVolume</u>	Provides an access to tick volumes of the bars in the history
<u>CiRealVolume</u>	Provides an access to real volumes of the bars in the history
<u>CiOpen</u>	Provides an access to open prices of the bars in the history
<u>CiHigh</u>	Provides an access to high prices of the bars in the history
<u>CiLow</u>	Provides an access to low prices of the bars in the history
<u>CiClose</u>	Provides an access to close prices of the bars in the history

CiSpread

CiSpread is a class designed for access to spreads of the bars in the history.

Description

CiSpread class provides an access to spread historical data.

Declaration

```
class CiSpread: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CiSpread

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets a series buffer size
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Create

Creates a timeseries with the specified parameters for access to the spreads history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

BufferResize

Sets size of the buffer series.

```
virtual void BufferResize(  
    int size    // size  
)
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified element of timeseries buffer.

```
int  GetData(  
    int  index      // index  
    ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(  
    int  start_pos,    // position  
    int  count,        // number  
    int& buffer        // array  
    ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing the data.

Return Value

>=0 - successful, -1 - cannot receive the data.

GetData

Gets data from a timeseries buffer by initial time and number.

```
int  GetData(  
    datetime start_time,    // starting time  
    int      count,        // number  
    int&     buffer        // array  
    ) const
```

Parameters

start_time

[in] Time of the timeseries buffer's initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by start and stop times.

```
int  GetData(  
    datetime  start_time,    // starting time  
    datetime  stop_time,     // stop time  
    int&      buffer         // array  
) const
```

Parameters

start_time

[in] Starting time of a timeseries buffer element.

stop_time

[in] Stop time of a timeseries buffer element.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive the data.

Refresh

Updates the data of timeseries.

```
virtual void Refresh(  
    int flags    // flags  
)
```

Parameters

flags

[in] Timeframe flags.

CiTime

CiTime is a class designed for access to open times of the bars in the history.

Description

CiTime class provides an access to open times of the bars in the history.

Declaration

```
class CiTime: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CiTime

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets timeseries buffer size
Data Access	
GetData	Gets the timeseries data
Data Update	
Refresh	Updates the timeseries data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Create

Creates a timeseries with the specified parameters for access to the opening times of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create timeseries.

BufferResize

Sets series buffer sizes.

```
virtual void BufferResize(  
    int size    // size  
)
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified element of a timeseries buffer.

```
datetime GetData(  
    int index      // index  
    ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the data from a timeseries buffer by starting position and number.

```
int GetData(  
    int start_pos,    // position  
    int count,        // number  
    long& buffer      // array  
    ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from timeseries buffer by starting time and number.

```
int GetData(  
    datetime start_time,    // starting time  
    int count,              // number  
    long& buffer            // array  
    ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting and stop times.

```
int  GetData(  
    datetime  start_time,    // starting time  
    datetime  stop_time,     // stop time  
    long&     buffer         // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer end element.

buffer

[in] Reference to the array for storing data

Return Value

>=0 - successful, -1 - cannot receive data.

Refresh

Updates the data of timeseries.

```
virtual void Refresh(  
    int flags    // flags  
)
```

Parameters

flags

[in] Timeframe flags.

CiTickVolume

CiTickVolume is a class designed for access to tick volumes of the bars in the history.

Description

CiTickVolume class provides an access to tick volumes of the bars in the history.

Declaration

```
class CiTickVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CiTickVolume

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets buffer sizes
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Create

Creates a timeseries with the specified parameters for access to the tick volumes of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period  // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

BufferResize

Sets size of the series buffer.

```
virtual void BufferResize(  
    int size    // size  
)
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified timeseries buffer element.

```
long  GetData(  
    int  index      // index  
    ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the data from the timeseries buffer by starting position and number.

```
int  GetData(  
    int  start_pos,    // position  
    int  count,        // number  
    long& buffer       // array  
    ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing the data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting time and number.

```
int  GetData(  
    datetime start_time, // starting time  
    int      count,      // number  
    long&    buffer      // array  
    ) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting and stop times.

```
int  GetData(  
    datetime  start_time,    // starting time  
    datetime  stop_time,     // stop time  
    long&     buffer         // array  
) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

stop_time

[in] Time of the timeseries buffer end element.

buffer

[in] Reference to the array for storing data

Return Value

>=0 - successful, -1 - cannot receive data.

Refresh

Updates the data of timeseries.

```
virtual void Refresh(  
    int flags    // flags  
)
```

Parameters

flags

[in] Timeframe flags.

CiRealVolume

CiRealVolume is a class designed for access to real volumes of the bars in the history.

Description

CiRealVolume class provides an access to real volumes of the bars in the history.

Declaration

```
class CiRealVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

CiRealVolume

Class Methods by Groups

Create	
Create	Creates a series
BufferResize	Sets buffer size
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Create

Creates a timeseries with the specified parameters for access to the real volumes of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

BufferResize

Sets a series buffer size.

```
virtual void BufferResize(  
    int size    // size  
)
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified series buffer element.

```
datetime GetData(  
    int index      // index  
    ) const
```

Parameters

index

[in] Buffer element index.

Return Value

Series buffer element, or 0.

GetData

Gets the data from timeseries buffer by starting position and number.

```
int GetData(  
    int start_pos,    // position  
    int count,        // number  
    long& buffer      // array  
    ) const
```

Parameters

start_pos

[in] Starting position of timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from timeseries buffer by starting time and number.

```
int GetData(  
    datetime start_time,    // starting time  
    int count,              // number  
    long& buffer            // array  
    ) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(  
    datetime  start_time,    // starting time  
    datetime  stop_time,     // stop time  
    long&     buffer         // target array  
) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Return Value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void Refresh(  
    int flags    // flags  
)
```

Parameters

flags

[in] Timeframe flags.

CiOpen

CiOpen is a class designed for access to open prices of the bars in the history.

Description

CiOpen class provides an access to open prices of the bars in the history.

Declaration

```
class CiOpen: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CPriceSeries](#)

CiOpen

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

Create

Creates a timeseries with the specified parameters for access to the open prices of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

GetData

Gets the element of timeseries by starting position and number.

```
int  GetData(  
    int    start_pos,      // starting position  
    int    count,         // number  
    double& buffer        // array  
) const
```

Parameters

start_pos

[in] Starting position of timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting time and number.

```
int  GetData(  
    datetime start_time,   // starting time  
    int      count,       // number  
    double&  buffer       // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from the timeseries buffer by starting and stop times.


```
int  GetData(  
    datetime  start_time,      // starting time  
    datetime  stop_time,       // stop time  
    double&    buffer          // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

CiHigh

CiHigh is a class designed for access to high prices of the bars in the history.

Description

CiHigh class provides an access to high prices of the bars in the history.

Declaration

```
class CiHigh: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CPriceSeries](#)

CiHigh

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

Create

Creates a timeseries with the specified parameters for access to the high prices of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

GetData

Gets data from the timeseries buffer by starting position and number.

```
int  GetData(  
    int    start_pos,      // position  
    int    count,         // number  
    double& buffer        // array  
) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting time and number.

```
int  GetData(  
    datetime start_time,   // starting time  
    int      count,       // number  
    double&  buffer       // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting and stop times.


```
int  GetData(  
    datetime  start_time,      // starting time  
    datetime  stop_time,       // stop time  
    double&   buffer           // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

CiLow

CiLow is a class designed for access to low prices of the bars in the history.

Description

CiLow class provides an access to low prices of the bars in the history.

Declaration

```
class CiLow: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CPriceSeries](#)

[CiLow](#)

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets a series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

Create

Creates a timeseries with the specified parameters for access to the low prices of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(  
    int    start_pos,      // position  
    int    count,         // number  
    double& buffer        // array  
) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting time and number.

```
int  GetData(  
    datetime start_time,   // starting time  
    int      count,        // number  
    double&  buffer        // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting and stop times.


```
int  GetData(  
    datetime  start_time,      // starting time  
    datetime  stop_time,       // stop time  
    double&   buffer           // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array

Return Value

>=0 - successful, -1 - cannot receive data.

CiClose

CiClose is a class designed for access to close prices of the bars in the history.

Description

CiClose class provides an access to close prices of the bars in the history.

Declaration

```
class CiClose: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CPriceSeries](#)

CiClose

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets series data

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

Create

Creates a timeseries with the specified parameters for access to the closing prices of the bars in the history.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

GetData

Gets data from a timeseries buffer by starting position and number.

```
int  GetData(  
    int    start_pos,      // position  
    int    count,         // number  
    double& buffer        // array  
) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from a timeseries buffer by starting time and number.

```
int  GetData(  
    datetime start_time,   // starting time  
    int      count,        // number  
    double&  buffer        // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of a timeseries buffer element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from a timeseries buffer by starting and stop times.


```
int  GetData(  
    datetime  start_time,      // starting time  
    datetime  stop_time,       // stop time  
    double&    buffer          // array  
) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

Main and Auxiliary Classes of Technical Indicators and Timeseries

This group of chapters contains technical details of the main and auxiliary classes of technical indicators and timeseries, as well as descriptions of the appropriate components of the MQL5 Standard Library.

Class/group	Description
<u>CiADX</u>	Average Directional Index
<u>CiADXWilder</u>	Average Directional Index by Welles Wilder
<u>CiBands</u>	Bollinger Bands®
<u>CiEnvelopes</u>	Envelopes
<u>CiIchimoku</u>	Ichimoku Kinko Hyo
<u>CiMA</u>	Moving Average
<u>CiSAR</u>	Parabolic Stop And Reverse System
<u>CiStdDev</u>	Standard Deviation
<u>CiDEMA</u>	Double Exponential Moving Average
<u>CiTEMA</u>	Triple Exponential Moving Average
<u>CiFrAMA</u>	Fractal Adaptive Moving Average
<u>CiAMA</u>	Adaptive Moving Average
<u>CiVIDyA</u>	Variable Index Dynamic Average

CiADX

CiADX is a class intended for using the Average Directional Index technical indicator.

Description

CiADX class provides the creation, configuration, and access to the data of the Average Directional Index indicator.

Declaration

```
class CiADX: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiADX

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element of the main line
Plus	Returns the buffer element of the +DI line
Minus	Returns the buffer element of the -DI line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element of the main line by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Main line buffer element index.

Return Value

Main line buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double Plus(  
    int index    // index  
)
```

Parameters

index

[in] +DI line buffer element index.

Return Value

The buffer element of the +DI line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double Minus (  
    int index    // index  
)
```

Parameters

index

[in] -DI line buffer element index.

Return Value

The buffer element of the -DI line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ADX](#) for CiADX).

CiADXWilder

CiADXWilder is a class intended for using the technical indicator Average Directional Index by Welles Wilder.

Description

CiADXWilder class provides the creation, configuration, and access to the data of the Average Directional Index by Welles Wilder.

Declaration

```
class CiADXWilder: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiADXWilder

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Plus	Returns the buffer data of the +DI line
Minus	Returns the buffer data of the -DI line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element of the main line by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Main line buffer element index.

Return Value

Main line buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double Plus(  
    int index    // index  
)
```

Parameters

index

[in] +DI line buffer element index.

Return Value

The buffer element of the +DI line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double Minus (  
    int index    // index  
)
```

Parameters

index

[in] -DI line buffer element index.

Return Value

The buffer element of the -DI line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ADXW](#) for CiADWWilder).

CiBands

CiBands is a class intended for using the Bollinger Bands® technical indicator.

Description

CiBands class provides the creation, configuration, and access to the data of the Bollinger Bands indicator.

Declaration

```
class CiBands: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiBands

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Base	Returns the buffer element of the base line
Upper	Returns the buffer element of the upper line
Lower	Returns the buffer element of the lower line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift.

```
int MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Deviation

Returns the deviation.

```
double Deviation() const
```

Return Value

Returns the deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             ma_shift,         // shift  
    double          deviation,        // deviation  
    int             applied           // applied price, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift of the indicator.

deviation

[in] Deviation.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Base

Returns the buffer element of the base line by the specified index.

```
double Base(  
    int index    // index  
)
```

Parameters

index

[in] Base line buffer element index.

Return Value

The buffer element of the base line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Upper

Returns the buffer element of the upper line by the specified index.

```
double Upper (  
    int index    // index  
)
```

Parameters

index

[in] Upper line buffer element index.

Return Value

The buffer element of the upper line of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double Lower (  
    int index    // index  
)
```

Parameters

index

[in] Lower line buffer element index.

Return Value

The buffer element of the lower line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_BANDS](#) for CiBands).

CiEnvelopes

CiEnvelopes is a class intended for using the Envelopes technical indicator.

Description

CiEnvelopes class provides the creation, configuration, and access to the data of the Envelopes indicator.

Declaration

```
class CiEnvelopes: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiEnvelopes

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Upper	Returns the buffer data of the upper line
Lower	Returns the buffer data of the lower line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation ([ENUM_MA_METHOD](#) enumeration value).

Deviation

Returns the value of deviation.

```
double Deviation() const
```

Return Value

Returns the value of deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,       // averaging period  
    int             ma_shift,        // shift  
    ENUM_MA_METHOD  ma_method,       // averaging method  
    int             applied,         // price type, handle  
    double          deviation        // deviation  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Price axis shift.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Object (price type or handle) to apply.

deviation

[in] Deviation.

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the buffer element of the upper line by the specified index.

```
double Upper (  
    int index    // index  
)
```

Parameters

index

[in] Upper line buffer element index.

Return Value

The buffer element of the upper line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double Lower (  
    int index    // index  
)
```

Parameters

index

[in] Lower line buffer element index.

Return Value

The buffer element of the lower line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ENVELOPES](#) for CiEnvelopes).

Cilchimoku

Cilchimoku is a class intended for using the Ichimoku Kinko Hyo technical indicator.

Description

Cilchimoku class provides the creation, setup, and access to the data of the Ichimoku Kinko Hyo indicator.

Declaration

```
class CiIchimoku: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

Cilchimoku

Class Methods by Groups

Attributes	
TenkanSenPeriod	Returns the TenkanSen period
KijunSenPeriod	Returns the KijunSen period
SenkouSpanBPeriod	Returns the SenkouSpanB period
Create	
Create	Creates the indicator
Data Access	
TenkanSen	Returns the buffer element of the TenkanSen line
KijunSen	Returns the buffer element of the KijunSen line
SenkouSpanA	Returns the buffer element of the SenkouSpanA line
SenkouSpanB	Returns the buffer element of the SenkouSpanB line

ChinkouSpan	Returns the buffer element of the ChinkouSpan line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), Redrawer, [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

TenkanSenPeriod

Returns the TenkanSen period.

```
int TenkanSenPeriod() const
```

Return Value

Returns the TenkanSen period, defined at the indicator creation.

KijunSenPeriod

Returns the KijunSen period.

```
int KijunSenPeriod() const
```

Return Value

Returns the KijunSen period, defined at the indicator creation.

SenkouSpanBPeriod

Returns the SenkouSpanB period.

```
int SenkouSpanBPeriod() const
```

Return Value

Returns the SenkouSpanB period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,           // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             tenkan_sen,       // period of TenkanSen  
    int             kijun_sen,        // period of KijunSen  
    int             senkou_span_b    // period of SenkouSpanB  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

tenkan_sen

[in] Period of TenkanSen.

kijun_sen

[in] Period of KijunSen.

senkou_span_b

[in] Period of SenkouSpanB.

Return Value

true - successful, false - cannot create the indicator.

TenkanSen

Returns the buffer element of the TenkanSen line by the specified index.

```
double TenkanSen(  
    int index    // index  
)
```

Parameters

index

[in] TenkanSen line buffer element index.

Return Value

The buffer element of the TenkanSen line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

KijunSen

Returns the buffer element of the KijunSen line by the specified index.

```
double KijunSen(  
    int index    // index  
)
```

Parameters

index

[in] KijunSen line buffer element index.

Return Value

The buffer element of the KijunSen line of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

SenkouSpanA

Returns the buffer element of the SenkouSpanA line by the specified index.

```
double SenkouSpanA(  
    int index    // index  
)
```

Parameters

index

[in] SenkouSpanA line buffer element index.

Return Value

The buffer element of the SenkouSpanA line of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

SenkouSpanB

Returns the buffer element of the SenkouSpanB line by the specified index.

```
double SenkouSpanB(  
    int index    // index  
)
```

Parameters

index

[in] SenkouSpanB line buffer element index.

Return Value

The buffer element of the SenkouSpanB line of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

ChinkouSpan

Returns the buffer element of the ChinkouSpan line by the specified index.

```
double ChinkouSpan(  
    int index    // index  
)
```

Parameters

index

[in] ChinkouSpan line buffer element index.

Return Value

The buffer element of the ChinkouSpan line of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ICHIMOKU](#) for Cilchimoku).

CiMA

CiMA is a class intended for using the Moving Average technical indicator.

Description

CiMA class provides the creation, setup, and access to the data of the Moving Average indicator.

Declaration

```
class CiMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

```

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiMA
  
```

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method (value of [ENUM_MA_METHOD](#) enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             ma_shift,         // shift  
    ENUM_MA_METHOD  ma_method,        // averaging method  
    int             applied           // price type, handle  
)
```

Parameters

string

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_MA](#) for CiMA).

CiSAR

CiSAR is a class intended for using the Parabolic Stop And Reverse System technical indicator.

Description

CiSAR class provides the creation, setup, and access to the data of the Parabolic Stop And Reverse System indicator.

Declaration

```
class CiSAR: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiSAR

Class Methods by Groups

Attributes	
SarStep	Returns the step of price increment
Maximum	Returns the maximum value of the step
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

SarStep

Returns the step of price increment.

```
double SarStep() const
```

Return Value

The step of price increment, defined at the indicator creation.

Maximum

Returns the maximum value of the step.

```
double Maximum() const
```

Return Value

The maximum value of the step, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period,      // period  
    double          step,        // step  
    double          maximum      // coefficient  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

step

[in] Step for the velocity increasing.

maximum

[in] Price following coefficient.

Return Value

true - successful, false - cannot change the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_SAR](#) for CiSAR).

CiStdDev

CiStdDev is a class intended for using the Standard Deviation technical indicator.

Description

CiStdDev class provides the creation, setup, and access to the data of the Standard Deviation indicator.

Declaration

```
class CiStdDev: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiStdDev

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method (value of [ENUM_MA_METHOD](#) enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,       // averaging period  
    int             ma_shift,        // shift  
    ENUM_MA_METHOD  ma_method,       // averaging method  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index if successful, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_STDDEV](#) for CiStdDev).

CiDEMA

CiDEMA is a class intended for using the Double Exponential Moving Average technical indicator.

Description

CiDEMA class provides the creation, setup, and access to the data of the Double Exponential Moving Average indicator.

Declaration

```
class CiDEMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiDEMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int IndShift () const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          string,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,       // averaging period  
    int             ind_shift,       // shift  
    int             applied          // price type, handle  
)
```

Parameters

string

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_DEMA](#) for CiDEMA).

CiTEMA

CiTEMA is a class intended for using the Triple Exponential Moving Average technical indicator.

Description

CiTEMA class provides the creation, setup, and access to the data of the Triple Exponential Moving Average indicator.

Declaration

```
class CiTEMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

CObject

CArray

CArrayObj

CSeries

CIndicator

CiTEMA

Class Methods by Groups

Attributes	
<u>MaPeriod</u>	Returns the averaging period
<u>IndShift</u>	Returns the horizontal shift
<u>Applied</u>	Returns the price type or handle to apply
Create	
<u>Create</u>	Creates the indicator
Data Access	
<u>Main</u>	Returns the buffer data
Input/output	
virtual <u>Type</u>	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int IndShift () const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             ma_shift,         // shift  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_TEMA](#) for CiTEMA).

CiFrAMA

CiFrAMA is a class intended for using the Fractal Adaptive Moving Average technical indicator.

Description

CiFrAMA class provides the creation, setup, and access to the data of the Fractal Adaptive Moving Average indicator.

Declaration

```
class CiFrAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiFrAMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int IndShift () const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             ma_shift,         // shift  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index if successful, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_FRAMA](#) for CiFrAMA).

CiAMA

CiAMA is a class intended for using the Adaptive Moving Average technical indicator.

Description

CiAMA class provides the creation, setup, and access to the data of the Adaptive Moving Average indicator.

Declaration

```
class CiAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiAMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
FastEmaPeriod	Returns the averaging period for the fast EMA
SlowEmaPeriod	Returns the averaging period for the slow EMA
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int IndShift () const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          string,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             fast_ema_period,  // fast EMA period  
    int             slow_ema_period,  // slow EMA period  
    int             ind_shift,        // shift  
    int             applied           // price type, handle  
)
```

Parameters

string

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_AMA](#) for CiAMA).

CiVIDyA

CiVIDyA is a class intended for using the Variable Index Dynamic Average technical indicator.

Description

CiVIDyA class provides the creation, setup, and access to the data of the Variable Index Dynamic Average indicator.

Declaration

```
class CiVIDyA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiVIDyA

Class Methods by Groups

Attributes	
CmoPeriod	Returns the period for Momentum
EmaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

CmoPeriod

Closes the chart bound to the class instance.

```
int CmoPeriod() const
```


EmaPeriod

Returns the chart symbol.

```
int EmaPeriod() const
```

Return Value

Returns a chart symbol bound to a class instance. "" - no bound chart.

IndShift

Returns the horizontal shift of the indicator.

```
int IndShift () const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             cmo_period,      // momentum period  
    int             ema_period,      // averaging period  
    int             ind_shift,       // shift  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

cmo_period

[in] Momentum period.

ema_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_VIDYA](#) for CiVIDyA).

Oscillator Classes

This group of chapters contains the technical details of Oscillators classes, as well as descriptions of the appropriate components of the MQL5 Standard Library.

Class/group	Description
<u>CiATR</u>	Average True Range
<u>CiBearsPower</u>	Bears Power
<u>CiBullsPower</u>	Bulls Power
<u>CiCCI</u>	Commodity Channel Index
<u>CiChaikin</u>	Chaikin Oscillator
<u>CiDeMarker</u>	DeMarker
<u>CiForce</u>	Force Index
<u>CiMACD</u>	Moving Averages Convergence-Divergence
<u>CiMomentum</u>	Momentum
<u>CiOsMA</u>	Moving Average of Oscillator (MACD histogram)
<u>CiRSI</u>	Relative Strength Index
<u>CiRVI</u>	Relative Vigor Index
<u>CiStochastic</u>	Stochastic Oscillator
<u>CiWPR</u>	Williams' Percent Range
<u>CiTriX</u>	Triple Exponential Moving Averages Oscillator

CiATR

CiATR is a class intended for using the Average True Range technical indicator.

Description

CiATR class provides the creation, setup, and access to the data of the Average True Range indicator.

Declaration

```
class CiATR: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

```

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiATR
  
```

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ATR](#) for CiATR).

CiBearsPower

CiBearsPower is a class intended for using the Bears Power technical indicator.

Description

CiBearsPower class provides the creation, setup, and access to the data of the Bears Power indicator.

Declaration

```
class CiBearsPower: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiBearsPower

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_BEARS](#) for CiBearsPower).

CiBullsPower

CiBullsPower is a class intended for using the Bulls Power technical indicator.

Description

CiBullsPower class provides the creation, setup, and access to the data of the Bulls Power indicator.

Declaration

```
class CiBullsPower: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiBullsPower

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_BULLS](#) for CiBullsPower).

CiCCI

CiCCI is a class intended for using the Commodity Channel Index technical indicator.

Description

CiCCI class provides the creation, setup, and access to the data of the Commodity Channel Index indicator.

Declaration

```
class CiCCI: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiCCI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_CCI](#) for CiCCI).

CiChaikin

CiChaikin is a class intended for using the Chaikin Oscillator technical indicator.

Description

CiChaikin class provides the creation, setup, and access to the data of the Chaikin Oscillator indicator.

Declaration

```
class CiChaikin: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiChaikin

Class Methods by Groups

Attributes	
FastMaPeriod	Returns the averaging period for the fast MA
SlowMaPeriod	Returns the averaging period for the slow MA
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

FastMaPeriod

Returns the averaging period for the fast EMA.

```
int FastMaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowMaPeriod

Returns the averaging period for the slow EMA.

```
int SlowMaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation ([ENUM_MA_METHOD](#) enumeration value).

Applied

Returns the object (volume type) to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Object (volume type) to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             fast_ma_period,  // fast EMA period  
    int             slow_ma_period,  // slow EMA period  
    ENUM_MA_METHOD  ma_method,       // averaging method  
    ENUM_APPLIED_VOLUME applied      // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

fast_ma_period

[in] Period for fast EMA.

slow_ma_period

[in] Period for slow EMA.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Object (volume type) to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_CHAIKIN](#) for CiChaikin).

CiDeMarker

CiDeMarker is a class intended for using the DeMarker technical indicator.

Description

CiDeMarker class provides the creation, setup, and access to the data of the DeMarker indicator.

Declaration

```
class CiDeMarker: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiDeMarker

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string      symbol,           // symbol  
    ENUM_TIMEFRAMES period,       // period  
    int         ma_period         // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_DEMARKER](#) for CiDeMarker).

CiForce

CiForce is a class intended for using the Force Index technical indicator.

Description

CiForce class provides the creation, setup, and access to the data of the Force Index indicator.

Declaration

```
class CiForce: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiForce

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaMethod	Returns the averaging method
Applied	Returns the object (volume type) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the object (volume type) to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Object (volume type) to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    ENUM_MA_METHOD  ma_method,        // averaging method  
    ENUM_APPLIED_VOLUME applied       // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Object (volume type) to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_FORCE](#) for CiForce).

CiMACD

CiMACD is a class intended for using the Moving Averages Convergence-Divergence technical indicator.

Description

CiMACD class provides the creation, setup, and access to the data of the Moving Averages Convergence-Divergence indicator.

Declaration

```
class CiMACD: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiMACD

Class Methods by Groups

Attributes	
FastEmaPeriod	Returns the averaging period of the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int SignalPeriod() const
```

Return Value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,           // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             fast_ema_period,  // fast EMA period  
    int             slow_ema_period,  // slow EMA period  
    int             signal_period,    // signal period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double Signal(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_MACD](#) for CiMACD).

CiMomentum

CiMomentum is a class intended for using the Momentum technical indicator.

Description

CiMomentum class provides the creation, setup, and access to the data of the Momentum indicator.

Declaration

```
class CiMomentum: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiMomentum

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the object (volume type) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_MOMENTUM](#) for CiMomentum).

CiOsMA

CiOsMA is a class intended for using the Moving Average of Oscillator (MACD histogram) technical indicator.

Description

CiOsMA class provides the creation, setup, and access to the data of the Moving Average of Oscillator (MACD histogram) indicator.

Declaration

```
class CiOsMA: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiOsMA

Class Methods by Groups

Attributes	
FastEmaPeriod	Returns the averaging period of the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int SignalPeriod() const
```

Return Value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,           // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             fast_ema_period,  // fast EMA period  
    int             slow_ema_period,  // slow EMA period  
    int             signal_period,    // signal line period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_OSMA](#) for CiOsMA).

CiRSI

CiRSI is a class intended for using the Relative Strength Index technical indicator.

Description

CiRSI class provides the creation, setup, and access to the data of the Relative Strength Index indicator.

Declaration

```
class CiRSI: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiRSI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_RSI](#) for CiRSI).

CiRVI

CiRVI is a class intended for using the Relative Vigor Index technical indicator.

Description

CiRVI class provides the creation, setup, and access to the data of the Relative Vigor Index indicator.

Declaration

```
class CiRVI: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

```

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiRVI
  
```

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period        // averaging period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double Signal(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_RVI](#) for CiRVI).

CiStochastic

CiStochastic is a class intended for using the Stochastic Oscillator technical indicator.

Description

CiStochastic class provides the creation, setup, and access to the data of the Stochastic Oscillator indicator.

Declaration

```
class CiStochastic: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiStochastic

Class Methods by Groups

Attributes	
Kperiod	Returns the averaging period for the %K line
Dperiod	Returns the averaging period for the %D line
Slowing	Returns the slowing period
MaMethod	Returns the averaging method
PriceField	Price type (Low/High or Close/Close) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Kperiod

Returns the averaging period for the %K line.

```
int Kperiod() const
```

Return Value

Returns the averaging period for the %K line, defined at the indicator creation.

Dperiod

Returns the averaging period for the %D line.

```
int Dperiod() const
```

Return Value

Returns the averaging period for the %D line, defined at the indicator creation.

Slowing

Returns the period of slowing.

```
int Slowing() const
```

Return Value

Returns the period of slowing, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation ([ENUM_MA_METHOD](#) enumeration value).

PriceField

Returns the object (Low/High or Close/Close) to apply.

```
ENUM_STO_PRICE PriceField() const
```

Return Value

The object (Low/High or Close/Close) to apply, defined at the indicator creation ([ENUM_STO_PRICE](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             Kperiod,         // %K period  
    int             Dperiod,         // %D period  
    int             slowing,         // slowing period  
    ENUM_MA_METHOD  ma_method,       // averaging method  
    ENUM_STO_PRICE  price_field      // application  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Kperiod

[in] Averaging period of %K indicator.

Dperiod

[in] Averaging period of %D indicator.

slowing

[in] Slowing period.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

price_field

[in] Object (Low/High or Close/Close) to apply ([ENUM_STO_PRICE](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double Signal(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_STOCHASTIC](#) for CiStochastic).

CiTriX

CiTriX is a class intended for using the Triple Exponential Moving Averages Oscillator technical indicator.

Description

CiTriX class provides the creation, setup, and access to the data of the Triple Exponential Moving Averages Oscillator indicator.

Declaration

```
class CiTriX: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiTriX

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    int             applied           // price type, handle  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type of handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_TRIX](#) for CiTriX).

CiWPR

CiWPR is a class intended for using the Williams' Percent Range technical indicator.

Description

CiWPR class provides the creation, setup, and access to the data of the Williams' Percent Range indicator.

Declaration

```
class CiWPR: public CIndicator
```

Title

```
#include <Indicators\Oscillators.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiWPR

Class Methods by Groups

Attributes	
CalcPeriod	Returns the calculation period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

CalcPeriod

Returns the period for calculation.

```
int CalcPeriod() const
```

Return Value

Returns the period for calculation, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             calc_period      // calculation period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

calc_period

[in] Period for calculation.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_WPR](#) for CiWPR).

Volume Indicators

This group of chapters contains technical details of Volume indicator classes and descriptions of all the appropriate key components of the MQL5 Standard Library.

Class/group	Description
<u>CiAD</u>	Accumulation/Distribution
<u>CiMFI</u>	Money Flow Index
<u>CiOBV</u>	On Balance Volume
<u>CiVolumes</u>	Volumes

CiAD

CiAD is a class intended for using the Accumulation/Distribution technical indicator.

Description

CiAD class provides the creation, setup, and access to the data of the Accumulation/Distribution indicator.

Declaration

```
class CiAD: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiAD

Class Methods by Groups

Attributes	
Applied	Returns the calculation period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period,      // period  
    ENUM_APPLIED_VOLUME applied  // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

applied

[in] Volume type to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_AD](#) for CiAD).

CiMFI

CiMFI is a class intended for using the Money Flow Index technical indicator.

Description

CiMFI class provides the creation, setup, and access to the data of the Money Flow Index indicator.

Declaration

```
class CiMFI: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiMFI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the volume type to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

MaPeriod

Returns the averaging period.

```
int MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // period  
    int             ma_period,        // averaging period  
    ENUM_APPLIED_VOLUME applied       // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

ma_period

[in] Averaging period.

applied

[in] Volume type to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_MFI](#) for CiMFI).

CiOBV

CiOBV is a class intended for using the On Balance Volume technical indicator.

Description

CiOBV class provides the creation, setup, and access to the data of the On Balance Volume indicator.

Declaration

```
class CiOBV: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

```

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiOBV

```

Class Methods by Groups

Attributes	
Applied	Returns the volume type to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period,      // period  
    ENUM_APPLIED_VOLUME applied  // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

applied

[in] Volume type to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_OBV](#) for CiOBV).

CiVolumes

CiVolumes is a class intended for using the Volumes technical indicator.

Description

CiVolumes class provides the creation, setup, and access to the data of the Volumes indicator.

Declaration

```
class CiVolumes: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiVolumes

Class Methods by Groups

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period,      // period  
    ENUM_APPLIED_VOLUME applied  // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

applied

[in] Volume type to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_VOLUMES](#) for CiVolumes).

Bill Williams Indicators

This group of chapters contains technical details of Bill Williams indicator classes and descriptions of all the appropriate components of the MQL5 Standard Library.

Class/group	Description
<u>CiAC</u>	Accelerator Oscillator
<u>CiAlligator</u>	Alligator
<u>CiAO</u>	Awesome Oscillator
<u>CiFractals</u>	Fractals
<u>CiGator</u>	Gator Oscillator
<u>CiBWMFI</u>	Market Facilitation Index

CiAC

CiAC is a class intended for using the Accelerator Oscillator technical indicator.

Description

CiAC class provides the creation, setup, and access to the data of the Accelerator Oscillator indicator.

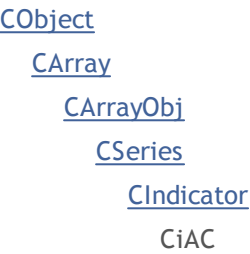
Declaration

```
class CiAC: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string      symbol,      // symbol  
    ENUM_TIMEFRAMES period    // period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_AC](#) for CiAC).

CiAlligator

CiAlligator is a class intended for using the Alligator technical indicator.

Description

CiAlligator class provides the creation, setup, and access to the data of the Alligator indicator.

Declaration

```
class CiAlligator: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiAlligator

Class Methods by Groups

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeth line
TeethShift	Returns the horizontal shift of the Teeth line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Jaw	Returns the buffer data of the Jaws line buffer
Teeth	Returns the buffer data of the Teeth line buffer

Lips	Returns the buffer data of the Lips line buffer
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), Redrawer, [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

JawPeriod

Returns the averaging period for the Jaw line.

```
int  JawPeriod() const
```

Return Value

Returns the averaging period for the Jaw line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift () const
```

Return Value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int TeethPeriod() const
```

Return Value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeth line.

```
int TeethShift() const
```

Return Value

Horizontal shift of the Teeth line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int LipsPeriod() const
```

Return Value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int LipsShift() const
```

Return Value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(
    string          symbol,           // symbol
    ENUM_TIMEFRAMES period,          // period
    int             jaw_period,       // jaws period
    int             jaw_shift,        // jaws shift
    int             teeth_period,     // teeth period
    int             teeth_shift,      // teeth shift
    int             lips_period,      // lips period
    int             lips_shift,       // lips shift
    ENUM_MA_METHOD  ma_method,       // averaging method
    int             applied           // price type, handle
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

jaw_period

[in] Jaws averaging period.

jaw_shift

[in] Jaws horizontal shift.

teeth_period

[in] Teeth averaging period.

teeth_shift

[in] Teeth horizontal shift.

lips_period

[in] Lips averaging period.

lips_shift

[in] Lips horizontal shift.

ma_method

[in] Moving average method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Price type, handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Jaw

Returns the buffer element of the Jaws line by the specified index.

```
double  Jaw(  
    int  index    // index  
)
```

Parameters

index

[in] Jaws line buffer element index.

Return Value

The buffer element of the Jaws line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Teeth

Returns the buffer element of the Teeth line by the specified index.

```
double Teeth(  
    int index    // index  
)
```

Parameters

index

[in] Teeth line buffer element index.

Return Value

The buffer element of the Teeth line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Lips

Returns the buffer element of the Lips line by the specified index.

```
double Lips(  
    int index    // index  
)
```

Parameters

index

[in] Lips line buffer element index.

Return Value

The buffer element of the Lips line by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_ALLIGATOR](#) for CiAlligator).

CiAO

CiAO is a class intended for using the Awesome Oscillator technical indicator.

Description

CiAO class provides the creation, setup, and access to the data of the Awesome Oscillator indicator.

Declaration

```
class CiAO: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, [Prev](#), Next, [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period      // period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_AO](#) for CiAO).

CiFractals

CiFractals is a class intended for using the Fractals technical indicator.

Description

CiFractals class provides the creation, setup, and access to the data of the Fractals indicator.

Declaration

```
class CiFractals: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

[CObject](#)

[CArray](#)

[CArrayObj](#)

[CSeries](#)

[CIndicator](#)

CiFractals

Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Upper	Returns the data of the upper buffer
Lower	Returns the data of the lower buffer
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),
[GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period      // period  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the element of the upper buffer by the specified index.

```
double Upper (  
    int index    // index  
)
```

Parameters

index

[in] Upper buffer element index.

Return Value

The element of the upper buffer by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Lower

Returns the element of the lower buffer by the specified index.

```
double Lower (  
    int index    // index  
)
```

Parameters

index

[in] Lower buffer element index.

Return Value

The element of the lower buffer by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_FRACTALS](#) for CiFractals).

CiGator

CiGator is a class intended for using the Gator Oscillator technical indicator.

Description

CiGator class provides the creation, setup, and access to the data of the Gator Oscillator indicator.

Declaration

```
class CiGator: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

```

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiGator
  
```

Class Methods by Groups

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeth line
TeethShift	Returns the horizontal shift of the Teeth line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Upper	Returns the data of the upper buffer
Lower	Returns the data of the lower buffer

Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Compare](#)

Methods inherited from class CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

JawPeriod

Returns the averaging period for the Jaws line.

```
int JawPeriod() const
```

Return Value

Returns the averaging period for the Jaws line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift () const
```

Return Value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int TeethPeriod() const
```

Return Value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeth line.

```
int TeethShift() const
```

Return Value

Horizontal shift of the Teeth line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int LipsPeriod() const
```

Return Value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int LipsShift() const
```

Return Value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(
    string          symbol,           // symbol
    ENUM_TIMEFRAMES period,          // period
    int             jaw_period,       // jaws period
    int             jaw_shift,        // jaws shift
    int             teeth_period,     // teeth period
    int             teeth_shift,      // teeth shift
    int             lips_period,      // lips period
    int             lips_shift,       // lips shift
    ENUM_MA_METHOD  ma_method,       // averaging method
    int             applied           // price type, handle
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

jaw_period

[in] Jaws averaging period.

jaw_shift

[in] Jaws horizontal shift.

teeth_period

[in] Teeth averaging period.

teeth_shift

[in] Teeth horizontal shift.

lips_period

[in] Lips averaging period.

lips_shift

[in] Lips horizontal shift.

ma_method

[in] Averaging method ([ENUM_MA_METHOD](#) enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the element of the upper buffer by the specified index.

```
double Upper (  
    int index    // index  
)
```

Parameters

index

[in] Upper buffer element index.

Return Value

The upper buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Lower

Returns the element of the lower buffer by the specified index.

```
double Upper (  
    int index    // index  
)
```

Parameters

index

[in] Lower buffer element index.

Return Value

The element of the lower buffer by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_GATOR](#) for CiGator).

CiBWMFI

CiBWMFI is a class intended for using the Market Facilitation Index by Bill Williams technical indicator.

Description

CiBWMFI class provides the creation, setup, and access to the data of the Market Facilitation Index by Bill Williams indicator.

Declaration

```
class CiBWMFI: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

CObject

CArray

CArrayObj

CSeries

CIndicator

CiBWMFI

Class Methods by Groups

Attributes	
<u>Applied</u>	Returns the volume type to apply
Create	
<u>Create</u>	Creates the indicator
Data Access	
<u>Main</u>	Returns the buffer data
Input/output	
virtual <u>Type</u>	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

Methods inherited from class CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Methods inherited from class CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation ([ENUM_APPLIED_VOLUME](#) enumeration value).

Create

Creates the indicator with specified parameters. Use [Refresh\(\)](#) and [GetData\(\)](#) to update and get the indicator values.

```
bool Create(  
    string          symbol,      // symbol  
    ENUM_TIMEFRAMES period,      // period  
    ENUM_APPLIED_VOLUME applied  // volume type  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration value).

applied

[in] Volume type to apply ([ENUM_APPLIED_VOLUME](#) enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double Main(  
    int index    // index  
)
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or [EMPTY_VALUE](#) if there is no correct data.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_BWMFI](#) for CiBWMFI).

CiCustom

CiCustom is a class intended for using the custom technical indicators.

Description

CiCustom class provides the creation, setup, and access to the data of a custom indicator.

Declaration

```
class CiCustom: public CIndicator
```

Title

```
#include <Indicators\Custom.mqh>
```

Class Methods by Groups

Attributes	
NumBuffers	Sets the number of buffers
NumParams	Gets the number of parameters used when creating an indicator
ParamType	Gets the type of the specified parameter
ParamLong	Gets the value of the specified parameter of integer type
ParamDouble	Gets the value of the specified parameter of double type
ParamString	Gets the value of the specified parameter of string type
Input/output	
virtual Type	Virtual identification method

NumBuffers

Sets the number of buffers.

```
bool NumBuffers (
```

Return Value

true - successful, false - cannot set the necessary number of buffers.

NumParams

Gets the number of parameters used when creating an indicator.

```
int NumParams() const
```

Return Value

Number of parameters, used in creation of the indicator.

ParamType

Gets a type of the parameter.

```
ENUM_DATATYPE ParamType(  
    int index    // index  
) const
```

Parameters

index

[in] Parameter index.

Return Value

Returns the data type of the specified parameter, used in indicator creation.

Note

If parameter index is invalid, it returns [WRONG_VALUE](#).

ParamLong

Gets the value of specified parameter of long type.

```
long ParamLong(  
    int index    // index  
    ) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified parameter of long type, used in creation of the indicator.

Note

If the parameter index is invalid or the parameter is not of long type, it returns 0.

ParamDouble

Gets the value of specified parameter of double type.

```
double ParamDouble(  
    int index    // index  
) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified parameter of double type, used in creation of the indicator.

Note

If the parameter index is invalid or the parameter type is not of double type, it returns [EMPTY_VALUE](#).

ParamString

Gets the value of specified parameter of string type.

```
string ParamString(  
    int index    // index  
    ) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified string parameter, used in creation of the indicator.

Note

If the number is invalid or the parameter is not of string type, it returns an empty string.

Type

Virtual identification method.

```
virtual int Type() const
```

Return Value

Indicator type ([IND_CUSTOM](#) for CiCustom).

Trade Classes

This section contains technical details of working with trade classes and description of the relevant components of the MQL5 standard library.

Using trade classes will save time when creating custom programs (Expert Advisors).

MQL5 Standard Library (in terms of trade classes) is placed in the terminal working directory, in the Include\Trade folder.

Class/Group	Description
<u>CAccountInfo</u>	Class for working with trade account properties
<u>CSymbolInfo</u>	Class for working with trade instrument properties
<u>COrderInfo</u>	Class for working with pending order properties
<u>CHistoryOrderInfo</u>	Class for working with history order properties
<u>CPositionInfo</u>	Class for working with open position properties
<u>CDealInfo</u>	Class for working with history deal properties
<u>CTrade</u>	Class for trade operations execution
<u>CTerminalInfo</u>	Class for getting the properties of the terminal environment

CAccountInfo

CAccountInfo is a class for easy access to the currently opened trade account properties.

Description

CAccountInfo class provides easy access to the currently opened trade account properties.

Declaration

```
class CAccountInfo : public CObject
```

Title

```
#include <Trade\AccountInfo.mqh>
```

Inheritance hierarchy

CObject

CAccountInfo

Class methods by groups

Access to integer type properties	
<u>Login</u>	Gets the account number
<u>TradeMode</u>	Gets the trade mode
<u>TradeModeDescription</u>	Gets the trade mode as a string
<u>Leverage</u>	Gets the amount of given leverage
<u>StopoutMode</u>	Gets the mode of stop out setting
<u>StopoutModeDescription</u>	Gets the mode of stop out setting as a string
<u>TradeAllowed</u>	Gets the flag of trade allowance
<u>TradeExpert</u>	Gets the flag of automated trade allowance
<u>LimitOrders</u>	Gets the maximal number of allowed pending orders
<u>MarginMode</u>	Gets margin calculation mode
<u>MarginModeDescription</u>	Gets margin calculation mode as a string
Access to double type properties	
<u>Balance</u>	Gets the balance of account
<u>Credit</u>	Gets the amount of given credit
<u>Profit</u>	Gets the amount of current profit on account
<u>Equity</u>	Gets the amount of current equity on account

Margin	Gets the amount of reserved margin
FreeMargin	Gets the amount of free margin
MarginLevel	Gets the level of margin
MarginCall	Gets the level of margin for deposit
MarginStopOut	Gets the level of margin for Stop Out
Access to text properties	
Name	Gets the client name
Server	Gets the trade server name
Currency	Gets the deposit currency name
Company	Gets the company name that serves an account
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Additional methods	
OrderProfitCheck	Gets the evaluated profit based on the parameters passed
MarginCheck	Gets the amount of margin required to execute trade operation
FreeMarginCheck	Gets the amount of free margin left after execution of trade operation
MaxLotCheck	Gets the maximal possible volume of trade operation

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Login

Gets the account number.

```
long Login() const
```

Return Value

Account number.

TradeMode

Gets the trade mode.

```
ENUM_ACCOUNT_TRADE_MODE TradeMode() const
```

Return Value

Trade mode from the [ENUM_ACCOUNT_TRADE_MODE](#) enumeration.

TradeModeDescription

Gets the trade mode as a string.

```
string TradeModeDescription() const
```

Return Value

Trade mode as a string.

Leverage

Gets the amount of given leverage.

```
long Leverage() const
```

Return Value

Amount of given leverage.

StopoutMode

Gets the mode of the Stop Out level specification.

```
ENUM_ACCOUNT_STOPOUT_MODE StopoutMode() const
```

Return Value

The stop out setting mode from the [ENUM_ACCOUNT_STOPOUT_MODE](#) enumeration.

StopoutModeDescription

Gets the mode of the Stop Out level specification as a string.

```
string StopoutModeDescription() const
```

Return Value

The stop out setting mode as a string.

MarginMode

Gets the margin calculation mode.

```
ENUM_ACCOUNT_MARGIN_MODE MarginMode() const
```

Return Value

The margin calculation mode from the [ENUM_ACCOUNT_MARGIN_MODE](#) enumeration.

MarginModeDescription

Gets the margin calculation mode as a string.

```
string MarginModeDescription() const
```

Return Value

Margin calculation mode as a string.

TradeAllowed

Gets the flag of trade allowance.

```
bool TradeAllowed() const
```

Return Value

Flag of trade allowance.

TradeExpert

Gets the flag of automated trade allowance.

```
bool TradeExpert() const
```

Return Value

Flag of automated trade allowance.

LimitOrders

Gets the maximal number of allowed pending orders

```
int LimitOrders() const
```

Return Value

The maximal number of allowed pending orders.

Note

0 - no limits.

Balance

Gets the balance of account.

```
double Balance() const
```

Return Value

The balance of account (in deposit currency).

Credit

Gets the amount of given credit.

```
double Credit() const
```

Return Value

Amount of given credit (in deposit currency).

Profit

Gets the amount of current profit on account.

```
double Profit() const
```

Return Value

Amount of current profit on account (in deposit currency).

Equity

Gets the amount of current equity on account.

```
double Equity() const
```

Return Value

Amount of current equity on account (in deposit currency).

Margin

Gets the amount of reserved margin.

```
double Margin() const
```

Return Value

Amount of reserved margin (in deposit currency).

FreeMargin

Gets the amount of free margin.

```
double FreeMargin() const
```

Return Value

Amount of free margin (in deposit currency).

MarginLevel

Gets the level of margin.

```
double MarginLevel() const
```

Return Value

Level of margin.

MarginCall

Gets the level of margin for a deposit.

```
double MarginCall() const
```

Return Value

Level of margin for a deposit.

MarginStopOut

Gets the level of margin for Stop Out.

```
double MarginStopOut () const
```

Return Value

Level of margin for Stop Out.

Name

Gets the client name.

```
string Name() const
```

Return Value

Client name.

Server

Gets the trade server name.

```
string Server() const
```

Return Value

Trade server name.

Currency

Gets the deposit currency name.

```
string Currency() const
```

Return Value

Deposit currency name.

Company

Gets the company name, that serves an account.

```
string Company() const
```

Return Value

Company name that serves an account.

InfoInteger

Gets the value of specified integer type property.

```
long InfoInteger(  
    ENUM_ACCOUNT_INFO_INTEGER prop_id    // property ID  
) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_INTEGER](#) enumeration.

Return Value

Value of [long](#) type.

InfoDouble

Gets the value of specified double type property.

```
double InfoDouble(  
    ENUM_ACCOUNT_INFO_DOUBLE prop_id // property ID  
) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_DOUBLE](#) enumeration.

Return Value

Value of [double](#) type.

InfoString

Gets the value of specified string type property.

```
string InfoString(  
    ENUM_ACCOUNT_INFO_STRING prop_id    // property ID  
) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of [ENUM_ACCOUNT_INFO_STRING](#) enumeration.

Return Value

Value of [string](#) type.

OrderProfitCheck

The function calculates the profit for the current account, based on the parameters passed. The function is used for pre-evaluation of the result of a trade operation. The value is returned in the account currency.

```
double OrderProfitCheck(  
    const string      symbol,           // symbol  
    ENUM_ORDER_TYPE   trade_operation,  // order type (ORDER_TYPE_BUY or ORDER_TYP  
    double            volume,           // volume  
    double            price_open,       // position open price  
    double            price_close       // position close price  
) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from [ENUM_ORDER_TYPE](#) enumeration.

volume

[in] Volume of trade operation.

price_open

[in] Open price.

price_close

[in] Close price.

Return Value

If successful, it returns amount of profit or [EMPTY_VALUE](#) in the case of error.

MarginCheck

Gets the amount of margin, required for trade operation.

```
double MarginCheck(  
    const string      symbol,           // symbol  
    ENUM_ORDER_TYPE   trade_operation, // order type (ORDER_TYPE_BUY or ORDER_TYPE_SELL)  
    double            volume,          // volume  
    double            price            // price  
) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from [ENUM_ORDER_TYPE](#) enumeration.

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Return Value

Amount of margin required for trade operation.

FreeMarginCheck

Gets the amount of free margin left after trade operation.

```
double FreeMarginCheck(  
    const string      symbol,           // symbol  
    ENUM_ORDER_TYPE   trade_operation,  // order type (ORDER_TYPE_BUY or ORDER_TYPE_SELL)  
    double            volume,          // volume  
    double            price            // price  
) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from [ENUM_ORDER_TYPE](#) enumeration.

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Return Value

Amount of free margin left after trade operation.

MaxLotCheck

Gets the maximum possible volume of trade operation.

```
double MaxLotCheck(  
    const string      symbol,           // symbol  
    ENUM_ORDER_TYPE   trade_operation,  // order type (ORDER_TYPE_BUY or ORDER_TYP  
    double            price,           // price  
    double            percent=100      // percent of available margin (default is  
) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from [ENUM_ORDER_TYPE](#) enumeration.

price

[in] Price of trade operation.

percent=100

[in] Percent of available margin (in %) to be used for trade operation.

Return Value

Maximum possible volume of trade operation.

CSymbolInfo

CSymbolInfo is a class for easy access to the symbol properties.

Description

CSymbolInfo class provides access to the symbol properties.

Declaration

```
class CSymbolInfo : public CObject
```

Title

```
#include <Trade\SymbolInfo.mqh>
```

Inheritance hierarchy

[CObject](#)

CSymbolInfo

Class methods by groups

Controlling	
Refresh	Refreshes the symbol data
RefreshRates	Refreshes the symbol quotes
Properties	
Name	Gets/sets symbol name
Select	Gets/sets the "Market Watch" symbol flag
IsSynchronized	Checks the symbol synchronization with server
Volumes	
Volume	Gets the volume of last deal
VolumeHigh	Gets the maximal volume for a day
VolumeLow	Gets the minimal volume for a day
Miscellaneous	
Time	Gets the time of last quote
Spread	Gets the amount of spread (in points)
SpreadFloat	Gets the flag of floating spread
TicksBookDepth	Gets the depth of ticks saving
Levels	

<u>StopsLevel</u>	Gets the minimal indent for orders (in points)
<u>FreezeLevel</u>	Gets the distance of freezing trade operations (in points)
Bid prices	
<u>Bid</u>	Gets the current Bid price
<u>BidHigh</u>	Gets the maximal Bid price for a day
<u>BidLow</u>	Gets the minimal Bid price for a day
Ask prices	
<u>Ask</u>	Gets the current Ask price
<u>AskHigh</u>	Gets the maximal Ask price for a day
<u>AskLow</u>	Gets the minimal Ask price for a day
Prices	
<u>Last</u>	Gets the current Last price
<u>LastHigh</u>	Gets the maximal Last price for a day
<u>LastLow</u>	Gets the minimal Last price for a day
Trade modes	
<u>TradeCalcMode</u>	Gets the mode of contract cost calculation
<u>TradeCalcModeDescription</u>	Gets the mode of contract cost calculation as a string
<u>TradeMode</u>	Gets the type of order execution
<u>TradeModeDescription</u>	Gets the type of order execution as a string
<u>TradeExecution</u>	Gets the trade execution mode
<u>TradeExecutionDescription</u>	Gets the execution mode as a string
Swaps	
<u>SwapMode</u>	Gets the swap calculation mode
<u>SwapModeDescription</u>	Gets the swap calculation mode as a string
<u>SwapRollover3days</u>	Gets the day of triple swap charge
<u>SwapRollover3daysDescription</u>	Gets the day of triple swap charge as a string
Margins and flags	
<u>MarginInitial</u>	Gets the value of initial margin
<u>MarginMaintenance</u>	Gets the value of maintenance margin

<u>MarginLong</u>	Gets the rate of margin charging for long positions
<u>MarginShort</u>	Gets the rate of margin charging for short positions
<u>MarginLimit</u>	Gets the rate of margin charging for Limit orders
<u>MarginStop</u>	Gets the rate of margin charging for Stop orders
<u>MarginStopLimit</u>	Gets the rate of margin charging for StopLimit orders
<u>TradeTimeFlags</u>	Gets the flags of allowed expiration modes
<u>TradeFillFlags</u>	Gets the flags of allowed filling modes
Quantization	
<u>Digits</u>	Gets the number of digits after period
<u>Point</u>	Gets the value of one point
<u>TickValue</u>	Gets the tick value (minimal change of price)
<u>TickValueProfit</u>	Gets the calculated tick price for a profitable position
<u>TickValueLoss</u>	Gets the calculated tick price for a losing position
<u>TickSize</u>	Gets the minimal change of price
Contracts sizes	
<u>ContractSize</u>	Gets the amount of trade contract
<u>LotsMin</u>	Gets the minimal volume to close a deal
<u>LotsMax</u>	Gets the maximal volume to close a deal
<u>LotsStep</u>	Gets the minimal step of volume change to close a deal
<u>LotsLimit</u>	Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol
Swaps sizes	
<u>SwapLong</u>	Gets the value of long position swap
<u>SwapShort</u>	Gets the value of short position swap
Text properties	
<u>CurrencyBase</u>	Gets the name of symbol base currency

CurrencyProfit	Gets the profit currency name
CurrencyMargin	Gets the margin currency name
Bank	Gets the name of current quote source
Description	Gets the string description of symbol
Path	Gets the path in symbols tree
Symbol properties	
SessionDeals	Gets the number of deals in the current session
SessionBuyOrders	Gets the number of Buy orders at the moment
SessionSellOrders	Gets the number of Sell orders at the moment
SessionTurnover	Gets the summary of turnover of the current session
SessionInterest	Gets the summary of open interest of the current session
SessionBuyOrdersVolume	Gets the current volume of Buy orders
SessionSellOrdersVolume	Gets the current volume of Sell orders
SessionOpen	Gets the open price of the current session
SessionClose	Gets the close price of the current session
SessionAW	Gets the average weighted price of the current session
SessionPriceSettlement	Gets the settlement price of the current session
SessionPriceLimitMin	Gets the minimal price of the current session
SessionPriceLimitMax	Gets the maximal price of the current session
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Service functions	
NormalizePrice	Returns the value of price, normalized using the symbol properties

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Refresh

Refreshes the symbol data.

```
void Refresh()
```

Return Value

None.

Note

The symbol should be selected by [Name](#) method.

RefreshRates

Refreshes the symbol quotes data.

```
bool RefreshRates ()
```

Return Value

true - success, false - unable to refresh quotes.

Note

The symbol should be selected by [Name](#) method.

Name

Gets symbol name.

```
string Name() const
```

Return Value

Symbol name.

Name

Sets symbol name.

```
bool Name(string name)
```

Return Value

None.

Select

Gets the "Market Watch" symbol flag.

```
bool Select() const
```

Return Value

"Market Watch" symbol flag.

Select

Sets the "Market Watch" symbol flag.

```
bool Select()
```

Return Value

true - success, false - unable to change flag.

Note

The symbol should be selected by [Name](#) method.

IsSynchronized

Checks the symbol synchronization with server.

```
bool IsSynchronized() const
```

Return Value

true - if the symbol is synchronized with server, false - if not.

Note

The symbol should be selected by [Name](#) method.

Volume

Gets the volume of last deal.

```
long Volume() const
```

Return Value

Volume of last deal.

Note

The symbol should be selected by [Name](#) method.

VolumeHigh

Gets the maximal volume of the day.

```
long VolumeHigh() const
```

Return Value

Maximal volume of the day.

Note

The symbol should be selected by [Name](#) method.

VolumeLow

Gets the minimal volume of the day.

```
long VolumeLow() const
```

Return Value

Minimal volume of the day.

Note

The symbol should be selected by [Name](#) method.

Time

Gets the time of last quote.

```
datetime Time() const
```

Return Value

Time of last quote.

Note

The symbol should be selected by [Name](#) method.

Spread

Gets the amount of spread (in points).

```
int Spread() const
```

Return Value

Gets the amount of spread (in points).

Note

The symbol should be selected by [Name](#) method.

SpreadFloat

Gets the flag of floating spread.

```
bool SpreadFloat() const
```

Return Value

Flag of floating spread.

Note

The symbol should be selected by [Name](#) method.

TicksBookDepth

Gets the depth of ticks saving.

```
int TicksBookDepth() const
```

Return Value

Depth of ticks saving.

Note

The symbol should be selected by [Name](#) method.

StopsLevel

Gets the minimal stop level for orders (in points).

```
int StopsLevel() const
```

Return Value

Minimal stop level for orders (in points).

Note

The symbol should be selected by [Name](#) method.

FreezeLevel

Gets the freeze level (in points).

```
int FreezeLevel() const
```

Return Value

Distance of freeze level (in points).

Note

The symbol should be selected by [Name](#) method.

Bid

Gets the current Bid price.

```
double Bid() const
```

Return Value

Current Bid price.

Note

The symbol should be selected by [Name](#) method.

BidHigh

Gets the maximal Bid price of the day.

```
double BidHigh() const
```

Return Value

Maximal Bid price of the day.

Note

The symbol should be selected by [Name](#) method.

BidLow

Gets the minimal Bid price of the day.

```
double BidLow() const
```

Return Value

Minimal Bid price of the day.

Note

The symbol should be selected by [Name](#) method.

Ask

Gets the current Ask price.

```
double Ask() const
```

Return Value

Current Ask price.

Note

The symbol should be selected by [Name](#) method.

AskHigh

Gets the maximal Ask price for a day.

```
double AskHigh() const
```

Return Value

Maximal Ask price of the day.

Note

The symbol should be selected by [Name](#) method.

AskLow

Gets the minimal Ask price for a day.

```
double AskLow() const
```

Return Value

Minimal Ask price of the day.

Note

The symbol should be selected by [Name](#) method.

Last

Gets the current Last price.

```
double Last() const
```

Return Value

Current Last price.

LastHigh

Gets the maximal Last price of the day.

```
double LastHigh() const
```

Return Value

Maximal Last price of the day.

Note

The symbol should be selected by [Name](#) method.

LastLow

Gets the minimal Last price of the day.

```
double LastLow() const
```

Return Value

Minimal Last price of the day.

Note

The symbol should be selected by [Name](#) method.

TradeCalcMode

Gets the mode of contract cost calculation.

```
ENUM_SYMBOL_CALC_MODE TradeCalcMode() const
```

Return Value

Mode of contract cost calculation from [ENUM_SYMBOL_CALC_MODE](#) enumeration.

Note

The symbol should be selected by [Name](#) method.

TradeCalcModeDescription

Gets the mode of contract cost calculation as a string.

```
string TradeCalcModeDescription() const
```

Return Value

Mode of contract cost calculation as a string.

Note

The symbol should be selected by [Name](#) method.

TradeMode

Gets the order execution type.

```
ENUM_SYMBOL_TRADE_MODE TradeMode() const
```

Return Value

Order execution type from [ENUM_SYMBOL_TRADE_MODE](#) enumeration.

Note

The symbol should be selected by [Name](#) method.

TradeModeDescription

Gets the trade mode as a string.

```
string TradeModeDescription() const
```

Return Value

Trade mode as a string.

Note

The symbol should be selected by [Name](#) method.

TradeExecution

Gets the trade execution mode.

```
ENUM_SYMBOL_TRADE_EXECUTION TradeExecution() const
```

Return Value

Trade execution mode from [ENUM_SYMBOL_TRADE_EXECUTION](#) enumeration.

Note

The symbol should be selected by [Name](#) method.

TradeExecutionDescription

Gets the description of trade execution mode as a string.

```
string TradeExecutionDescription() const
```

Return Value

Trade execution mode as a string.

Note

The symbol should be selected by [Name](#) method.

SwapMode

Gets the swap calculation mode.

```
ENUM_SYMBOL_SWAP_MODE SwapMode() const
```

Return Value

Swap calculation mode from [ENUM_SYMBOL_SWAP_MODE](#) enumeration.

Note

The symbol should be selected by [Name](#) method.

SwapModeDescription

Gets the swap mode description as a string.

```
string SwapModeDescription() const
```

Return Value

Swap mode description as a string.

Note

The symbol should be selected by [Name](#) method.

SwapRollover3days

Gets the swap rollover day.

```
ENUM_DAY_OF_WEEK SwapRollover3days () const
```

Return Value

Swap rollover day from [ENUM_DAY_OF_WEEK](#) enumeration.

Note

The symbol should be selected by [Name](#) method.

SwapRollover3daysDescription

Gets the swap rollover day as a string.

```
string SwapRollover3daysDescription() const
```

Return Value

Swap rollover day as a string.

Note

The symbol should be selected by [Name](#) method.

MarginInitial

Gets the value of initial margin.

```
double MarginInitial ()
```

Return Value

Value of initial margin.

Note

It returns the amount of margin (in margin currency of instrument) that is charged from one lot. Used to check client's equity, when they enter the market.

The symbol should be selected by [Name](#) method.

MarginMaintenance

Gets the value of maintenance margin.

```
double MarginMaintenance()
```

Return Value

Value of maintenance margin.

Note

It returns the amount of margin (in margin currency of instrument) that is charged from one lot. Used to check client's equity, when the account state is changed. If the maintenance margin is equal to 0, then the initial margin is used.

The symbol should be selected by [Name](#) method.

MarginLong

Gets the rate of margin charging on long positons.

```
double MarginLong() const
```

Return Value

Rate of margin charging on long positons.

Note

The symbol should be selected by [Name](#) method.

MarginShort

Gets the rate of margin charging on short positons.

```
double MarginShort() const
```

Return Value

Rate of margin charging on short positons.

Note

The symbol should be selected by [Name](#) method.

MarginLimit

Gets the rate of margin charging on Limit orders.

```
double MarginLimit() const
```

Return Value

Rate of margin charging on Limit orders.

Note

The symbol should be selected by [Name](#) method.

MarginStop

Gets the rate of margin charging on Stop orders.

```
double MarginStop() const
```

Return Value

Rate of margin charging on Stop orders.

Note

The symbol should be selected by [Name](#) method.

MarginStopLimit

Gets the rate of margin charging on Stop Limit orders.

```
double MarginStopLimit() const
```

Return Value

Rate of margin charging on Stop Limit orders.

Note

The symbol should be selected by [Name](#) method.

TradeTimeFlags

Gets the flags of allowed expiration modes.

```
int TradeTimeFlags() const
```

Return Value

Flags of allowed expiration modes.

Note

The symbol should be selected by [Name](#) method.

TradeFillFlags

Gets the flags of allowed filling modes.

```
int TradeFillFlags() const
```

Return Value

Flags of allowed filling modes.

Note

The symbol should be selected by [Name](#) method.

Digits

Gets the number of digits after period.

```
int Digits() const
```

Return Value

The number of digits after period.

Note

The symbol should be selected by [Name](#) method.

Point

Gets the value of one point.

```
double Point () const
```

Return Value

Value of one point.

Note

The symbol should be selected by [Name](#) method.

TickValue

Gets the tick value (minimal change of price).

```
double TickValue() const
```

Return Value

Tick value (minimal change of price).

Note

The symbol should be selected by [Name](#) method.

TickValueProfit

Gets the calculated tick price for a profitable position.

```
double TickValueProfit() const
```

Return Value

The calculated tick price for a profitable position.

Note

The symbol should be selected by [Name](#) method.

TickValueLoss

Gets the calculated tick price for a losing position.

```
double TickValueLoss () const
```

Return Value

The calculated tick price for a losing position.

Note

The symbol should be selected by [Name](#) method.

TickSize

Gets the minimal change of price.

```
double TickSize() const
```

Return Value

Minimal change of price.

Note

The symbol should be selected by [Name](#) method.

ContractSize

Gets the amount of trade contract.

```
double ContractSize() const
```

Return Value

Amount of trade contract.

Note

The symbol should be selected by [Name](#) method.

LotsMin

Gets the minimal volume to close a deal.

```
double LotsMin() const
```

Return Value

Minimal volume to close a deal.

Note

The symbol should be selected by [Name](#) method.

LotsMax

Gets the maximal volume to close a deal.

```
double LotsMax() const
```

Return Value

Maximal volume to close a deal.

Note

The symbol should be selected by [Name](#) method.

LotsStep

Gets the minimal step of volume change to close a deal.

```
double LotsStep() const
```

Return Value

Minimal step of volume change to close a deal.

Note

The symbol should be selected by [Name](#) method.

LotsLimit

Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

```
double LotsLimit() const
```

Return Value

The maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

Note

The symbol should be selected by [Name](#) method.

SwapLong

Gets the value of long position swap.

```
double SwapLong() const
```

Return Value

Value of long position swap.

Note

The symbol should be selected by [Name](#) method.

SwapShort

Gets the value of short position swap.

```
double SwapShort() const
```

Return Value

Value of short position swap.

Note

The symbol should be selected by [Name](#) method.

CurrencyBase

Gets the name of symbol base currency.

```
string CurrencyBase() const
```

Return Value

Name of symbol base currency.

Note

The symbol should be selected by [Name](#) method.

CurrencyProfit

Gets the profit currency name.

```
string CurrencyProfit() const
```

Return Value

Profit currency name.

Note

The symbol should be selected by [Name](#) method.

CurrencyMargin

Gets the margin currency name.

```
string CurrencyMargin() const
```

Return Value

Margin currency name.

Note

The symbol should be selected by [Name](#) method.

Bank

Gets the name of current quote source.

```
string Bank() const
```

Return Value

Name of current quote source.

Note

The symbol should be selected by [Name](#) method.

Description

Gets the string description of symbol.

```
string Description() const
```

Return Value

String description of symbol.

Note

The symbol should be selected by [Name](#) method.

Path

Gets the path in symbols tree.

```
string Path() const
```

Return Value

Gets the path in symbols tree.

Note

The symbol should be selected by [Name](#) method.

SessionDeals

Gets the number of deals in the current session.

```
long SessionDeals() const
```

Return Value

Number of deals in the current session.

Note

The symbol should be selected by [Name](#) method.

SessionBuyOrders

Gets the number of Buy orders at the moment.

```
long SessionBuyOrders() const
```

Return Value

Number of Buy orders at the moment.

Note

The symbol should be selected by [Name](#) method.

SessionSellOrders

Gets then number of Sell orders at the moment.

```
long SessionSellOrders() const
```

Return Value

Number of Sell orders at the moment.

Note

The symbol should be selected by [Name](#) method.

SessionTurnover

Gets summary turnover of the current session.

```
double SessionTurnover() const
```

Return Value

Summary turnover of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionInterest

Gets the summary of open interest of the current session.

```
double SessionInterest() const
```

Return Value

Summary open interest of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionBuyOrdersVolume

Gets the current volume of Buy orders.

```
double SessionBuyOrdersVolume () const
```

Return Value

Current volume of Buy orders.

Note

The symbol should be selected by [Name](#) method.

SessionSellOrdersVolume

Gets the current volume of Sell orders.

```
double SessionSellOrdersVolume() const
```

Return Value

Current volume of Sell orders.

Note

The symbol should be selected by [Name](#) method.

SessionOpen

Gets the open price of the current session.

```
double SessionOpen() const
```

Return Value

Open price of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionClose

Gets the close price of the current session.

```
double SessionClose() const
```

Return Value

Close price of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionAW

Gets the average weighted price of the current session.

```
double SessionAW() const
```

Return Value

Average weighted price of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionPriceSettlement

Gets the settlement price of the current session.

```
double SessionPriceSettlement () const
```

Return Value

Settlement price of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionPriceLimitMin

Gets the minimal price of the current session.

```
double SessionPriceLimitMin() const
```

Return Value

Minimal price of the current session.

Note

The symbol should be selected by [Name](#) method.

SessionPriceLimitMax

Gets the maximal price of the current session.

```
double SessionPriceLimitMax() const
```

Return Value

Maximal price of the current session.

Note

The symbol should be selected by [Name](#) method.

InfoInteger

Gets the value of specified integer type property.

```
bool InfoInteger (
    ENUM_SYMBOL_INFO_INTEGER prop_id,    // property ID
    long& var                             // reference to variable
) const
```

Parameters

prop_id

[in] ID of integer type property from [ENUM_SYMBOL_INFO_INTEGER](#) enumeration.

var

[out] Reference to [long](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by [Name](#) method.

InfoDouble

Gets the value of specified double type property.

```
bool InfoDouble(  
    ENUM_SYMBOL_INFO_DOUBLE prop_id,    // property ID  
    double& var                        // reference to variable  
) const
```

Parameters

prop_id

[in] ID of double type property from [ENUM_SYMBOL_INFO_DOUBLE](#) enumeration.

var

[out] Reference to [double](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by [Name](#) method.

InfoString

Gets the value of specified string type property.

```
bool InfoString(  
    ENUM_SYMBOL_INFO_STRING prop_id,    // property ID  
    string& var                        // reference to variable  
) const
```

Parameters

prop_id

[in] ID of text property.

var

[out] Reference to [string](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by [Name](#) method.

NormalizePrice

Returns the value of price normalized using the symbol properties.

```
double NormalizePrice(  
    double price // price  
    ) const
```

Parameters

price

[in] Price.

Return Value

Normalized price.

Note

The symbol should be selected by [Name](#) method.

COrderInfo

COrderInfo is a class for easy access to the pending order properties.

Description

COrderInfo class provides access to the pending order properties.

Declaration

```
class COrderInfo : public CObject
```

Title

```
#include <Trade\OrderInfo.mqh>
```

Inheritance hierarchy

CObject

COrderInfo

Class methods by groups

Access to integer type properties	
<u>Ticket</u>	Gets the ticket of an order, previously selected for access
<u>TimeSetup</u>	Gets the time of order placement
<u>TimeSetupMsc</u>	Receives the time of placing an order in milliseconds since 01.01.1970
<u>OrderType</u>	Gets the order type
<u>OrderTypeDescription</u>	Gets the order type as a string
<u>State</u>	Gets the order state
<u>StateDescription</u>	Gets the order state as a string
<u>TimeExpiration</u>	Gets the time of order expiration
<u>TimeDone</u>	Gets the time of order execution or cancellation
<u>TimeDoneMsc</u>	Receives order execution or cancellation time in milliseconds since 01.01.1970
<u>TypeFilling</u>	Gets the type of order execution by remainder
<u>TypeFillingDescription</u>	Gets the type of order execution by remainder as a string
<u>TypeTime</u>	Gets the type of order at the time of the expiration

TypeTimeDescription	Gets the order type by expiration time as a string
Magic	Gets the ID of expert that placed the order
PositionId	Gets the ID of position
Access to double type properties	
VolumeInitial	Gets the initial volume of order
VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of a Limit order
Access to text properties	
Symbol	Gets the name of order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
State	
StoreState	Saves the order parameters
CheckState	Checks the current parameters against the saved parameters
Selection	
Select	Selects an order by ticket for further access to its properties
SelectByIndex	Selects an order by index for further access to its properties

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Ticket

Gets the ticket of an order.

```
ulong Ticket () const
```

Return Value

Order ticket if successful, otherwise - [ULONG_MAX](#).

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeSetup

Gets the time of order placement.

```
datetime TimeSetup() const
```

Return Value

Time of order placement.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeSetupMsc

Receives the time of placing an order for execution in milliseconds since 01.01.1970.

```
ulong TimeSetupMsc() const
```

Return Value

The time of placing an order for execution in milliseconds since 01.01.1970.

Note

Order should be preliminarily selected for access using [Select](#) (by ticket) or [SelectByIndex](#) (by index) method.

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE OrderType()
```

Return Value

Order type from [ENUM_ORDER_TYPE](#) enumeration.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeDescription

Gets the order type as a string.

```
string TypeDescription() const
```

Return Value

Order type as a string.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE State() const
```

Return Value

Order state from [ENUM_ORDER_STATE](#) enumeration.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StateDescription

Gets the order state as a string.

```
string StateDescription() const
```

Return Value

Order state as a string.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeExpiration

Gets the order expiration time.

```
datetime TimeExpiration() const
```

Return Value

Order expiration time, set on its placement.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime TimeDone() const
```

Return Value

Time of order execution or cancellation.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeDoneMsc

Receives order execution or cancellation time in milliseconds since 01.01.1970.

```
ulong TimeDoneMsc() const
```

Return Value

Order execution or cancellation time in milliseconds since 01.01.1970.

Note

Order should be preliminarily selected for access using [Select](#) (by ticket) or [SelectByIndex](#) (by index) method.

TypeFilling

Gets the order filling type.

```
ENUM_ORDER_TYPE_FILLING TypeFilling() const
```

Return Value

Order filling type from [ENUM_ORDER_TYPE_FILLING](#) enumeration.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeFillingDescription

Gets the order filling type as a string.

```
string TypeFillingDescription() const
```

Return Value

Order filling type as a string.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME TypeTime () const
```

Return Value

Type of order at the time of the expiration from [ENUM_ORDER_TYPE_TIME](#) enumeration.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string TypeTimeDescription() const
```

Return Value

Order type by expiration time as a string.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Magic

Gets the ID of an Expert Advisor that placed the order.

```
long Magic() const
```

Return Value

ID of an Expert Advisor that placed the order.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PositionId

Gets the ID of position.

```
long PositionId() const
```

Return Value

ID of position, in which the order was involved.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double VolumeInitial () const
```

Return Value

Initial volume of order.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double VolumeCurrent() const
```

Return Value

Unfilled volume of order.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceOpen

Gets the order price.

```
double PriceOpen() const
```

Return Value

Price of order placement.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StopLoss

Gets the order's Stop Loss.

```
double StopLoss() const
```

Return Value

Order's Stop Loss.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TakeProfit

Gets the order's Take Profit.

```
double TakeProfit() const
```

Return Value

Order's Take Profit.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceCurrent

Gets the current price by order symbol.

```
double PriceCurrent() const
```

Return Value

Current price by order symbol.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceStopLimit

Gets the price of a pending order.

```
double PriceStopLimit() const
```

Return Value

Pending order price.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Symbol

Gets the name of order symbol.

```
string Symbol() const
```

Return Value

Name of order symbol.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Comment

Gets the order comment.

```
string Comment() const
```

Return Value

Order comment.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool InfoInteger(  
    ENUM_ORDER_PROPERTY_INTEGER prop_id,    // property ID  
    long& var                                // reference to variable  
) const
```

Parameters

prop_id

[in] ID of integer type property from [ENUM_ORDER_PROPERTY_INTEGER](#) enumeration.

var

[out] Reference to [long](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble (
    ENUM_ORDER_PROPERTY_DOUBLE prop_id,    // property ID
    double&          var                   // reference to variable
) const
```

Parameters

prop_id

[in] ID of double type property from [ENUM_ORDER_PROPERTY_DOUBLE](#) enumeration.

var

[out] Reference to [double](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool InfoString(  
    ENUM_ORDER_PROPERTY_STRING prop_id,    // property ID  
    string& var                            // reference to variable  
) const
```

Parameters

prop_id

[in] ID of string property from [ENUM_ORDER_PROPERTY_STRING](#) enumeration.

var

[out] Reference to [string](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StoreState

Saves the order parameters.

```
void StoreState()
```

Return Value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool CheckState()
```

Return Value

true - if the order parameters have changed since the last call of the [StoreState\(\)](#) method, otherwise
- false.

Select

Selects an order by ticket for further access to its properties.

```
bool Select(  
    ulong ticket // order ticket  
)
```

Return Value

true - success, false - unable to select order.

SelectByIndex

Selects an order by index.

```
bool SelectByIndex(  
    int index // order index  
)
```

Parameters

index

[in] Order index.

Return Value

true - success, false - unable to select order.

CHistoryOrderInfo

CHistoryOrderInfo is a class for easy access to the history order properties.

Description

CHistoryOrderInfo class provides easy access to the history order properties.

Declaration

```
class CHistoryOrderInfo : public CObject
```

Title

```
#include <Trade\HistoryOrderInfo.mqh>
```

Inheritance hierarchy

CObject

CHistoryOrderInfo

Class methods by groups

Access to integer type properties	
<u>TimeSetup</u>	Gets the time of order placement
<u>TimeSetupMsc</u>	Receives the time of placing an order in milliseconds since 01.01.1970
<u>OrderType</u>	Gets the order type
<u>OrderTypeDescription</u>	Gets the order type as a string
<u>State</u>	Gets the order state
<u>StateDescription</u>	Gets the order state as a string
<u>TimeExpiration</u>	Gets the time of order expiration
<u>TimeDone</u>	Gets the time of order execution or cancellation
<u>TimeDoneMsc</u>	Receives order execution or cancellation time in milliseconds since 01.01.1970
<u>TypeFilling</u>	Gets the type of order execution by remainder
<u>TypeFillingDescription</u>	Gets the type of order execution by remainder as a string
<u>TypeTime</u>	Gets the type of order at the time of the expiration
<u>TypeTimeDescription</u>	Gets the order type by expiration time as a string

Magic	Gets the ID of expert that placed the order
PositionId	Gets the ID of position
Access to double type properties	
VolumeInitial	Gets the initial volume of order
VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of a Limit order
Access to text properties	
Symbol	Gets the order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets the ticket/selects the order
SelectByIndex	Selects the order by index

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

TimeSetup

Gets the time of order placement.

```
datetime TimeSetup() const
```

Return Value

Time of order placement.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeSetupMsc

Receives the time of placing an order for execution in milliseconds since 01.01.1970.

```
ulong TimeSetupMsc() const
```

Return Value

The time of placing an order for execution in milliseconds since 01.01.1970.

Note

Historical order should be preliminarily selected for access using [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) method.

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE OrderType() const
```

Return Value

Order type from [ENUM_ORDER_TYPE](#) enumeration.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeDescription

Gets the order type as a string.

```
string TypeDescription() const
```

Return Value

Order type as a string.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE State() const
```

Return Value

Order state from [ENUM_ORDER_STATE](#) enumeration.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StateDescription

Gets the order state as a string.

```
string StateDescription() const
```

Return Value

Order state as a string.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeExpiration

Gets the time of order expiration.

```
datetime TimeExpiration() const
```

Return Value

Time of order expiration, set on its placement.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime TimeDone() const
```

Return Value

Time of order execution or cancellation.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeDoneMsc

Receives order execution or cancellation time in milliseconds since 01.01.1970.

```
ulong TimeDoneMsc() const
```

Return Value

Order execution or cancellation time in milliseconds since 01.01.1970.

Note

Historical order should be preliminarily selected for access using [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) method.

TypeFilling

Gets the type of order execution by remainder.

```
ENUM_ORDER_TYPE_FILLING TypeFilling() const
```

Return Value

Type of order execution by remainder from [ENUM_ORDER_TYPE_FILLING](#) enumeration.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeFillingDescription

Gets the type of order execution by remainder as a string.

```
string TypeFillingDescription() const
```

Return Value

Type order of execution by remainder as a string.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME TypeTime () const
```

Return Value

Type of order at the time of the expiration from [ENUM_ORDER_TYPE_TIME](#) enumeration.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string TypeTimeDescription() const
```

Return Value

Order type by expiration time as a string.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Magic

Gets the ID of the Expert Advisor, that placed the order.

```
long Magic() const
```

Return Value

ID of the Expert Advisor, that placed the order.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PositionId

Gets the ID of position.

```
long PositionId() const
```

Return Value

ID of position, in which the order was involved.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double VolumeInitial () const
```

Return Value

Initial volume of order.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double VolumeCurrent () const
```

Return Value

Unfilled volume of order.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceOpen

Gets the order price.

```
double PriceOpen() const
```

Return Value

Price of order placement.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StopLoss

Gets the Stop Loss price of the order.

```
double StopLoss() const
```

Return Value

Stop Loss price of the order.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TakeProfit

Gets the Take Profit price of the order.

```
double TakeProfit() const
```

Return Value

The Take Profit price of the order.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceCurrent

Gets the current price of the order's symbol.

```
double PriceCurrent() const
```

Return Value

The current price of order's symbol.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceStopLimit

Gets the pending order price.

```
double PriceStopLimit() const
```

Return Value

Pending orders price.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Symbol

Gets the name of order symbol.

```
string Symbol() const
```

Return Value

Name of order symbol.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Comment

Gets the order comment.

```
string Comment() const
```

Return Value

Order comment.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool InfoInteger (
    ENUM_ORDER_PROPERTY_INTEGER prop_id,    // property ID
    long& var                                // reference to variable
) const
```

Parameters

prop_id

[in] ID of integer type property from [ENUM_ORDER_PROPERTY_INTEGER](#) enumeration.

var

[out] Reference to [long](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool InfoDouble(  
    ENUM_ORDER_PROPERTY_DOUBLE prop_id,    // property ID  
    double& var                            // reference to variable  
) const
```

Parameters

prop_id

[in] ID of double type property from [ENUM_ORDER_PROPERTY_DOUBLE](#) enumeration.

var

[out] Reference to [double](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool InfoString(  
    ENUM_ORDER_PROPERTY_STRING prop_id,    // property ID  
    string& var                            // reference to variable  
) const
```

Parameters

prop_id

[in] ID of text property from [ENUM_ORDER_PROPERTY_STRING](#) enumeration.

var

[out] Reference to [string](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Ticket (Get method)

Gets the order ticket.

```
ulong Ticket() const
```

Return Value

Order ticket.

Ticket (Set method)

Select the order for further work.

```
void Ticket(  
    ulong ticket    // ticket  
)
```

Parameters

ticket

[in] Order ticket.

SelectByIndex

Selects a historical order by index.

```
bool SelectByIndex(  
    int index // order index  
)
```

Parameters

index

[in] Historical order index.

Return Value

true - success, false - unable to select order.

CPositionInfo

CPositionInfo is a class for easy access to the open position properties.

Description

CPositionInfo class provides easy access to the open position properties.

Declaration

```
class CPositionInfo : public CObject
```

Title

```
#include <Trade\PositionInfo.mqh>
```

Inheritance hierarchy

[CObject](#)

CPositionInfo

Class methods by groups

Access to integer type properties	
Time	Gets the time of position opening
TimeMsc	Receives the time of position opening in milliseconds since 01.01.1970
TimeUpdate	Receives the time of position changing in seconds since 01.01.1970
TimeUpdateMsc	Receives the time of position changing in milliseconds since 01.01.1970
PositionType	Gets the position type
TypeDescription	Gets the position type as a string
Magic	Gets the ID of expert, that opened the position
Identifier	Gets the ID of position
Access to double type properties	
Volume	Gets the volume of position
PriceOpen	Gets the price of position opening
StopLoss	Gets the price of position's Stop Loss
TakeProfit	Gets the price of position's Take Profit
PriceCurrent	Gets the current price by position symbol
Commission	Gets the amount of commission by position

Swap	Gets the amount of swap by position
Profit	Gets the amount of current profit by position
Access to text properties	
Symbol	Gets the name of position symbol
Comment	Gets the comment of the position
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Selection	
Select	Selects the position
SelectByIndex	Selects the position by index
SelectByMagic	Selects a position with the specified symbol name and magic number
SelectByTicket	Selects the position by ticket
State	
StoreState	Saves the position parameters
CheckState	Checks the current parameters against the saved parameters

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Time

Gets the time of position opening.

```
datetime Time() const
```

Return Value

Time of position opening.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeMsc

Receives position opening time in milliseconds since 01.01.1970.

```
ulong TimeMsc() const
```

Return Value

Position opening time in milliseconds since 01.01.1970.

Note

Position should be preliminarily selected for access using [Select](#) (by symbol) or [SelectByIndex](#) (by index) method.

TimeUpdate

Receives the time of position changing in seconds since 01.01.1970.

```
datetime TimeUpdate() const
```

Return Value

Time of position changing in seconds since 01.01.1970.

Note

Position should be preliminarily selected for access using [Select](#) (by symbol) or [SelectByIndex](#) (by index) method.

TimeUpdateMsc

Receives the time of position changing in milliseconds since 01.01.1970.

```
ulong TimeUpdateMsc() const
```

Return Value

The time of position changing in milliseconds since 01.01.1970.

Note

Position should be preliminarily selected for access using [Select](#) (by symbol) or [SelectByIndex](#) (by index) method.

PositionType

Gets the position type.

```
ENUM_POSITION_TYPE PositionType() const
```

Return Value

Position type from [ENUM_POSITION_TYPE](#) enumeration.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeDescription

Gets the position type as a string.

```
string TypeDescription() const
```

Return Value

Position type as a string.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Magic

Gets the ID of Expert Advisor that opened the position.

```
long Magic() const
```

Return Value

ID of the Expert Advisor that opened the position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Identifier

Gets the ID of position.

```
long Identifier() const
```

Return Value

ID of position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Volume

Gets the volume of position.

```
double Volume() const
```

Return Value

Volume of position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceOpen

Gets the price of position opening.

```
double PriceOpen() const
```

Return Value

Position open price.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

StopLoss

Gets the Stop Loss price of the position.

```
double StopLoss() const
```

Return Value

The Stop Loss price of the position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TakeProfit

Gets the Take Profit price of the position.

```
double TakeProfit() const
```

Return Value

The Take Profit price of the position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PriceCurrent

Gets the current price by position symbol.

```
double PriceCurrent() const
```

Return Value

Current price by position symbol.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Commission

Gets the amount of commission of the position.

```
double Commission() const
```

Return Value

Amount of commission of the position (in deposit currency).

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Swap

Gets the amount of swap of the position.

```
double Swap() const
```

Return Value

Amount of swap of the position (in deposit currency).

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Profit

Gets the amount of current profit of the position.

```
double Profit() const
```

Return Value

Amount of current profit of the position (in deposit currency).

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Symbol

Gets the name of position symbol.

```
string Symbol() const
```

Return Value

Name of position symbol.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Comment

Gets the comment of the position.

```
string Comment() const
```

Return Value

Comment of the position.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool InfoInteger(  
    ENUM_POSITION_PROPERTY_INTEGER prop_id,    // property ID  
    long& var                                // reference to variable  
) const
```

Parameters

prop_id

[in] ID of integer type property from [ENUM_POSITION_PROPERTY_INTEGER](#) enumeration.

var

[out] Reference to [long](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool InfoDouble(  
    ENUM_POSITION_PROPERTY_DOUBLE prop_id,    // property ID  
    double& var                               // reference to variable  
) const
```

Parameters

prop_id

[in] ID of double type property from [ENUM_POSITION_PROPERTY_DOUBLE](#) enumeration.

var

[in] Reference to [double](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool InfoString(  
    ENUM_POSITION_PROPERTY_STRING prop_id,    // property ID  
    string& var                               // reference to variable  
) const
```

Parameters

prop_id

[in] ID of text property from [ENUM_POSITION_PROPERTY_STRING](#) enumeration.

var

[out] Reference to [string](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the [Select](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Select

Select the position for further work.

```
bool Select(  
    const string symbol      // symbol  
)
```

Parameters

symbol

[in] Symbol for position selection.

SelectByIndex

Selects the position by index for further access to its properties.

```
bool SelectByIndex(  
    int index // position index  
);
```

Return Value

true - success, false - unable to select position.

SelectByMagic

Select a position based on the name of a financial instrument and magic number to further work with.

```
bool SelectByMagic(  
    const string symbol, // symbol name  
    const ulong magic    // magic number  
);
```

Parameters

symbol

[in] Symbol name.

magic

[in] Magic number of the position.

Return Value

true - successful, false - unable to select position.

SelectByTicket

Selects position by ticket for further operation.

```
bool SelectByTicket(  
    ulong ticket    // position ticket  
);
```

Parameters

ticket

[in] Position ticket.

Return value

true - success, false - no position selected.

StoreState

Saves the position parameters.

```
void StoreState()
```

Return Value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool CheckState()
```

Return Value

true - the position parameters have changed since the last call of the [StoreState\(\)](#) method,
otherwise - false.

CDealInfo

CDealInfo is a class for easy access to the deal properties.

Description

CDealInfo class provides access to the deal properties.

Declaration

```
class CDealInfo : public CObject
```

Title

```
#include <Trade\DealInfo.mqh>
```

Inheritance hierarchy

CObject

CDealInfo

Class methods by groups

Access to integer type properties	
<u>Order</u>	Gets the order by which the deal is executed
<u>Time</u>	Gets the time of deal execution
<u>TimeMsc</u>	Receives the time of a deal execution in milliseconds since 01.01.1970
<u>DealType</u>	Gets the deal type
<u>TypeDescription</u>	Gets the deal type as a string
<u>Entry</u>	Gets the deal direction
<u>EntryDescription</u>	Gets the deal direction as a string
<u>Magic</u>	Gets the ID of expert, that executed the deal
<u>PositionId</u>	Gets the ID of position, in which the deal was involved
Access to double type properties	
<u>Volume</u>	Gets the volume of deal
<u>Price</u>	Gets the deal price
<u>Commision</u>	Gets the amount of commission of the deal
<u>Swap</u>	Gets the amount of swap when position is closed
<u>Profit</u>	Gets the financial result of deal

Access to text properties	
Symbol	Gets the name of deal symbol
Comment	Gets the deal comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets ticket/selects the deal
SelectByIndex	Selects the deal by index

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Order

Gets the order by which the deal is executed.

```
long Order() const
```

Return Value

Order by which the deal is executed.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Time

Gets the time of deal execution.

```
datetime Time() const
```

Return Value

Time of deal execution.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TimeMsc

Receives the time of a deal execution in milliseconds since 01.01.1970.

```
ulong TimeMsc() const
```

Return Value

The time of a deal execution in milliseconds since 01.01.1970.

Note

Deal should be preliminarily selected for access using [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) method.

DealType

Gets the deal type.

```
ENUM_DEAL_TYPE DealType() const
```

Return Value

Deal type from [ENUM_DEAL_TYPE](#) enumeration.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

TypeDescription

Gets the deal type as a string.

```
string TypeDescription() const
```

Return Value

Deal type as a string.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Entry

Compares the deal direction.

```
ENUM_DEAL_ENTRY Entry() const
```

Return Value

Deal direction (value of [ENUM_DEAL_ENTRY](#) enumeration).

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

EntryDescription

Gets the deal direction as a string.

```
string EntryDescription() const
```

Return Value

Deal direction as a string.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Magic

Gets the ID of the Expert Advisor, that executed the deal.

```
long Magic() const
```

Return Value

ID of the Expert Advisor, that executed the deal.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

PositionId

Gets the ID of position, in which the deal was involved.

```
long PositionId() const
```

Return Value

ID of position, in which the deal was involved.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Volume

Gets the volume of deal.

```
double Volume() const
```

Return Value

Volume of deal.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Price

Gets the deal price.

```
double Price() const
```

Return Value

Deal price.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Commission

Gets the amount of commission of the deal.

```
double Commission() const
```

Return Value

Amount of commission of the deal.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Swap

Gets the amount of swap when position is closed.

```
double Swap() const
```

Return Value

Amount of swap when position is closed.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Profit

Gets the financial result of the deal.

```
double Profit() const
```

Return Value

Financial result of the deal (in deposit currency).

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Symbol

Gets the name of the deal symbol.

```
string Symbol() const
```

Return Value

Name of the deal symbol.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Comment

Gets the deal comment.

```
string Comment() const
```

Return Value

Deal comment.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool InfoInteger (
    ENUM_DEAL_PROPERTY_INTEGER prop_id,    // property ID
    long& var                               // reference to variable
) const
```

Parameters

prop_id

[in] ID of integer type property from [ENUM_DEAL_PROPERTY_INTEGER](#) enumeration.

var

[out] Reference to [long](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool InfoDouble(  
    ENUM_DEAL_PROPERTY_DOUBLE prop_id,    // property ID  
    double& var                          // reference to variable  
) const
```

Parameters

prop_id

[in] ID of double type property from [ENUM_DEAL_PROPERTY_DOUBLE](#) enumeration.

var

[out] Reference to [double](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool InfoString(  
    ENUM_DEAL_PROPERTY_STRING prop_id,    // property ID  
    string& var                            // reference to variable  
) const
```

Parameters

prop_id

[in] ID of text property from [ENUM_DEAL_PROPERTY_STRING](#) enumeration.

var

[out] Reference to [string](#) type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the [Ticket](#) (by ticket) or [SelectByIndex](#) (by index) methods.

Ticket (Get method)

Gets the deal ticket.

```
ulong Ticket() const
```

Return Value

Deal ticket.

Ticket (Set method)

Select the position for further work.

```
void Ticket(  
    ulong ticket    // ticket  
)
```

Parameters

ticket

[in] Deal ticket.

SelectByIndex

Selects the deal by index for further access to its properties.

```
bool SelectByIndex(  
    int index // order index  
)
```

Return Value

true - success, false - unable to select the deal.

CTrade

CTrade is a class for easy access to the trade functions.

Description

CTrade class provides easy access to the trade functions.

Declaration

```
class CTrade : public CObject
```

Title

```
#include <Trade\Trade.mqh>
```

Inheritance hierarchy

[CObject](#)

CTrade

Direct descendants

CExpertTrade

Class methods by groups

Setting parameters	
LogLevel	Sets logging level
SetExpertMagicNumber	Sets the expert ID
SetDeviationInPoints	Sets the allowed deviation
SetTypeFilling	Sets filling type of the order
SetAsyncMode	Sets asynchronous mode for trade operations
SetMarginMode	Sets margin calculation mode in accordance with the current account settings
Operations with orders	
OrderOpen	Places a pending order with specified parameters
OrderModify	Modifies the pending order parameters
OrderDelete	Deletes a pending order
Operations with positions	
PositionOpen	Opens a position with specified parameters
PositionModify	Modifies position parameters by the specified symbol or position ticket

<u>PositionClose</u>	Closes a position for the specified symbol
<u>PositionClosePartial</u>	Partially closes a position on a specified symbol or having a specified ticket
<u>PositionCloseBy</u>	Closes a position with the specified ticket by an opposite position
Additional methods	
<u>Buy</u>	Opens a long position with specified parameters
<u>Sell</u>	Opens a short position with specified parameters
<u>BuyLimit</u>	Places a pending order of the Buy Limit type with specified parameters
<u>BuyStop</u>	Places a pending order of the Buy Stop type with specified parameters
<u>SellLimit</u>	Places a pending order of the Sell Limit type with specified parameters
<u>SellStop</u>	Places a pending order of the Sell Stop type with specified parameters
Access to the last request parameters	
<u>Request</u>	Gets the copy of the last request structure
<u>RequestAction</u>	Gets the trade operation type
<u>RequestActionDescription</u>	Gets the trade operation type as string
<u>RequestMagic</u>	Gets the magic number of the Expert Advisor
<u>RequestOrder</u>	Gets the order ticket used in the last request
<u>RequestSymbol</u>	Gets the name of the symbol used in the last request
<u>RequestVolume</u>	Gets the trade volume (in lots) used in the last request
<u>RequestPrice</u>	Gets the price used in the last request
<u>RequestStopLimit</u>	Gets the price of pending order of Stop Limit type used in the last request
<u>RequestSL</u>	Gets the Stop Loss price of the order used in the last request
<u>RequestTP</u>	Gets the Take Profit price of the order used in the last request
<u>RequestDeviation</u>	Gets the maximum allowable price deviation of the order used in the last request

<u>RequestType</u>	Gets the type of the order used in the last request
<u>RequestTypeDescription</u>	Gets the type of the order (as string) used in the last request
<u>RequestTypeFilling</u>	Gets the filling type of the order used in the last request
<u>RequestTypeFillingDescription</u>	Gets the filling type of the order (as string) used in the last request
<u>RequestTypeTime</u>	Gets the validity period of the order used in the last request
<u>RequestTypeTimeDescription</u>	Gets the validity period of the order (as string) used in the last request
<u>RequestExpiration</u>	Gets the expiration time of the order used in the last request
<u>RequestComment</u>	Gets the comment of the order used in the last request
<u>RequestPosition</u>	Gets position ticket
<u>RequestPositionBy</u>	Gets opposite position ticket
Access to the last request checking results	
<u>CheckResult</u>	Gets the copy of the structure of the last request check result.
<u>CheckResultRetcode</u>	Gets the value of the retcode field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultRetcodeDescription</u>	Gets the string description of the retcode field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultBalance</u>	Gets the value of the balance field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultEquity</u>	Gets the value of the equity field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultProfit</u>	Gets the value of the floating profit after executing a trading operation.
<u>CheckResultMargin</u>	Gets the value of the margin field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness

<u>CheckResultMarginFree</u>	Gets the value of the margin_free field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultMarginLevel</u>	Gets the value of the margin_level field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
<u>CheckResultComment</u>	Gets the value of the comment field of <u>MqlTradeCheckResult</u> type, filled while checking the request correctness
Access to the last request execution results	
<u>Result</u>	Gets the copy of the structure of the last request result
<u>ResultRetcode</u>	Gets the code of request result
<u>ResultRetcodeDescription</u>	Gets the code of request result as a string
<u>ResultDeal</u>	Gets the deal ticket
<u>ResultOrder</u>	Gets the order ticket
<u>ResultVolume</u>	Gets the volume of deal or order
<u>ResultPrice</u>	Gets the price, confirmed by broker
<u>ResultBid</u>	Gets the current bid price (the requote)
<u>ResultAsk</u>	Gets the current ask price (the requote)
<u>ResultComment</u>	Gets the broker comment
Auxiliary methods	
<u>PrintRequest</u>	Prints the last request parameters into journal
<u>PrintResult</u>	Prints the results of the last request into journal
<u>FormatRequest</u>	Prepares the formatted string with last request parameters
<u>FormatRequestResult</u>	Prepares the formatted string with results of the last request execution

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

LogLevel

Sets logging level for messages.

```
void LogLevel(  
    ENUM_LOG_LEVELS log_level    // level  
)
```

Parameters

log_level

[in] New logging level.

Return Value

None.

Note

LOG_LEVEL_NO and less disables displaying any messages (set up automatically in the optimization mode). LOG_LEVEL_ERRORS enables displaying only error messages (value by default). LOG_LEVEL_ALL and greater enables displaying any messages (set up automatically in the test mode).

ENUM_LOG_LEVELS

Identifier	Description	Value
LOG_LEVEL_NO	Displaying messages disabled	0
LOG_LEVEL_ERRORS	Only error messages are displayed	1
LOG_LEVEL_ALL	All messages are displayed	2

SetExpertMagicNumber

Sets the expert ID.

```
void SetExpertMagicNumber (
    ulong magic    // ID
)
```

Parameters

magic

[in] New ID of the expert.

Return Value

None.

SetDeviationInPoints

Sets the allowed deviation.

```
void SetDeviationInPoints(  
    ulong deviation    // deviation  
)
```

Parameters

deviation

[in] Allowed deviation in points.

Return Value

None.

SetTypeFilling

Sets filling type of the order.

```
void SetTypeFilling(  
    ENUM_ORDER_TYPE_FILLING filling    // order filling type  
)
```

Parameters

filling

[in] Order filling type from [ENUM_ORDER_TYPE_FILLING](#) enumeration.

Return Value

None.

SetAsyncMode

Sets asynchronous mode for trade operations.

```
void SetAsyncMode(  
    bool mode    // asynchronous mode flag  
)
```

Parameters

mode

[in] Asynchronous mode flag.

Return Value

None.

Note

This mode is used for asynchronous (without waiting for the trade server's response to a sent request) trade operations (see [OrderSendAsync](#)).

SetMarginMode

Sets margin calculation mode in accordance with the current account settings.

```
void SetMarginMode ()
```

Return Value

No.

Note

The margin calculation mode is specified in [ENUM_ACCOUNT_MARGIN_MODE](#).

OrderOpen

Places the pending order with set parameters.

```
bool OrderOpen(
    const string      symbol,           // symbol
    ENUM_ORDER_TYPE   order_type,      // order type
    double            volume,          // order volume
    double            limit_price,      // StopLimit price
    double            price,           // execution price
    double            sl,              // Stop Loss price
    double            tp,              // Take Profit price
    ENUM_ORDER_TYPE_TIME type_time,     // type by expiration
    datetime          expiration,      // expiration
    const string      comment=""       // comment
)
```

Parameters

symbol

[in] Name of trade instrument.

order_type

[in] Type of order trade operation from [ENUM_ORDER_TYPE](#) enumeration.

volume

[in] Requested order volume.

limit_price

[in] Price at which the StopLimit order will be placed.

price

[in] Price at which the order must be executed.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

type_time

[in] Order type by execution from [ENUM_ORDER_TYPE_TIME](#) enumeration.

expiration

[in] Expiration date of pending order.

comment=""

[in] Order comment.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the `OrderSend(...)` method does not always mean successful execution of the trade operation. It is necessary to check the result of a trade request (trade server return code) using [ResultRetcode\(\)](#) and value returned by [ResultOrder\(\)](#).

OrderModify

Modifies the pending order parameters.

```
bool OrderModify(
    ulong          ticket,          // order ticket
    double         price,           // execution price
    double         sl,              // Stop Loss price
    double         tp,              // Take Profit price
    ENUM_ORDER_TYPE_TIME type_time, // type by expiration
    datetime       expiration,      // expiration
    double         stoplimit        // Limit order price
)
```

Parameters

ticket

[in] Order ticket.

price

[in] The new price by which the order must be executed (or the previous value, if the change is not necessary).

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

type_time

[in] The new type of order by expiration from [ENUM_ORDER_TYPE_TIME](#) enumeration (or the previous value, if the change is not necessary).

expiration

[in] The new expiration date of pending order (or the previous value, if the change is not necessary).

stoplimit

[in] New price used for setting a Limit order when the price reaches *price* value. It is specified only for StopLimit orders.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderModify(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using [ResultRetcode\(\)](#).

OrderDelete

Deletes the pending order.

```
bool OrderDelete(  
    ulong ticket    // order ticket  
)
```

Parameters

ticket

[in] Order ticket.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderDelete(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using [ResultRetcode\(\)](#).

PositionOpen

Opens a position with the specified parameters.

```
bool PositionOpen(  
    const string      symbol,           // symbol  
    ENUM_ORDER_TYPE  order_type,       // order type to open position  
    double            volume,           // position volume  
    double            price,            // execution price  
    double            sl,               // Stop Loss price  
    double            tp,               // Take Profit price  
    const string      comment=""        // comment  
)
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to open position.

order_type

[in] Order type (trade operation) to open position from [ENUM_ORDER_TYPE](#) enumeration.

volume

[in] Requested position volume.

price

[in] Price at which the position must be opened.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

comment=""

[in] Position comment.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the `PositionOpen(...)` method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using [ResultRetcode\(\)](#) and value returned by [ResultDeal\(\)](#).

PositionModify

Modifies the position parameters by specified symbol.

```
bool PositionModify(  
    const string  symbol,      // symbol  
    double        sl,         // Stop Loss price  
    double        tp          // Take Profit price  
)
```

Modifies position parameters by the specified ticket.

```
bool PositionModify(  
    const ulong   ticket,      // position ticket  
    double        sl,         // Stop Loss price  
    double        tp          // Take Profit price  
)
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to modify position.

ticket

[in] Ticket of the position to be modified.

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionModify(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using [ResultRetcode\(\)](#).

For the "netting" interpretation of positions ([ACCOUNT MARGIN MODE RETAIL NETTING](#) and [ACCOUNT MARGIN MODE EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT MARGIN MODE RETAIL HEDGING](#)), multiple positions can be open for one symbol. In this case, PositionModify will modify the position with the lowest ticket.

PositionClose

Closes a position by the specified symbol.

```
bool PositionClose(  
    const string  symbol,           // symbol  
    ulong        deviation=ULONG_MAX // deviation  
)
```

Closes a position with the specified ticket.

```
bool PositionClose(  
    const ulong   ticket,           // position ticket  
    ulong        deviation=ULONG_MAX // deviation  
)
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to close position.

ticket

[in] Ticket of a closed position.

deviation=ULONG_MAX

[in] Maximal deviation from the current price (in points).

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionClose(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using [ResultRetcode\(\)](#).

For the "netting" interpretation of positions ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), only one [position](#) can exist for a [symbol](#) at any moment of time. This position is a result of one or more [deals](#). Do not confuse positions with valid [pending orders](#), which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be open for one symbol. In this case, PositionClose will close a position with the lowest ticket.

PositionClosePartial

Partially closes a position on a specified symbol in case of a "hedging" accounting.

```
bool PositionClosePartial(
    const string  symbol,           // symbol
    const double  volume,           // volume
    ulong         deviation=ULONG_MAX // deviation
)
```

Partially closes a position having a specified ticket in case of a "hedging" accounting.

```
bool PositionClosePartial(
    const ulong   ticket,           // position ticket
    const double  volume,           // volume
    ulong         deviation=ULONG_MAX // deviation
)
```

Parameters

symbol

[in] Name of a trading instrument, on which a position is closed partially. If a symbol (not a ticket) is specified for a position partial closing, the first detected position having a specified MagicNumber ([Expert Advisor ID](#)) on the symbol is selected. Therefore, it is sometimes better to use PositionClosePartial() with a specified position ticket.

volume

[in] Volume, by which a position should be decreased. If the value exceeds the volume of a partially closed position, it is closed in full. No position in the opposite direction is opened.

ticket

[in] Closed position ticket.

deviation=ULONG_MAX

[in] The maximum deviation from the current price (in points).

Return Value

true if the basic check of structures is successful, otherwise false.

Note

A successful completion of the PositionClosePartial(...) method does not always mean a successful execution of a trading operation. You should call the [ResultRetcode\(\)](#) method to check the result of a trade request (trade server return code).

In the "netting" system ([ACCOUNT_MARGIN_MODE_RETAIL_NETTING](#) and [ACCOUNT_MARGIN_MODE_EXCHANGE](#)), for each [symbol](#), at any given moment of time, only one [position](#) can be open, which is the result of one or more [deals](#). Do not confuse the current [pending orders](#) with positions that are also displayed on the "Trade" tab of the client terminal's "Toolbox" panel.

In case of the position representation ([ACCOUNT_MARGIN_MODE_RETAIL_HEDGING](#)), multiple positions can be opened on each symbol simultaneously. In this case, PositionClose closes a position having a least ticket.

PositionCloseBy

Closes a position with the specified ticket by an opposite position.

```
bool PositionCloseBy(  
    const ulong   ticket,           // position ticket  
    const ulong   ticket_by        // opposite position ticket  
)
```

Parameters

ticket

[in] Ticket of the closed position.

ticket_by

[in] Ticket of the opposite position used for closing.

Returned value

true - the basic check of structures is successful, otherwise - false.

Note

A successful completion of the `PositionCloseBy(...)` method does not always mean a successful execution of a trading operation. You should call the [ResultRetcode\(\)](#) method to check the result of a trade request (trade server return code).

Buy

Opens a long position with specified parameters.

```
bool Buy(  
    double      volume,           // position volume  
    const string symbol=NULL,     // symbol  
    double      price=0.0,        // execution price  
    double      sl=0.0,           // stop loss price  
    double      tp=0.0,           // take profit price  
    const string comment=""      // comment  
)
```

Parameters

volume

[in] Requested position volume.

symbol=NULL

[in] Position symbol. If it is not specified, the current symbol will be used.

price=0.0

[in] Price. If the price is not specified, the current market Ask price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the Buy(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultDeal\(\)](#).

Sell

Opens a short position with specified parameters.

```
bool Sell(  
    double      volume,           // position volume  
    const string symbol=NULL,     // symbol  
    double      price=0.0,        // execution price  
    double      sl=0.0,           // stop loss price  
    double      tp=0.0,           // take profit price  
    const string comment=""      // comment  
)
```

Parameters

volume

[in] Requested position volume.

symbol=NULL

[in] Position symbol. If the symbol is not specified, the current symbol will be used.

price=0.0

[in] Price. If the price is not specified, the current market Bid price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the Sell(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultDeal\(\)](#).

BuyLimit

Places the pending order of Buy Limit type (buy at the price lower than current market price) with specified parameters.

```
bool BuyLimit(
    double          volume,           // order volume
    double          price,           // order price
    const string    symbol=NULL,     // symbol
    double          sl=0.0,          // stop loss price
    double          tp=0.0,          // take profit price
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // order lifetime
    datetime        expiration=0,    // order expiration time
    const string    comment=""       // comment
)
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from [ENUM_ORDER_TYPE_TIME](#) enumeration.

expiration=0

[in] Order expiration time (used only if type_time=[ORDER_TIME_SPECIFIED](#)).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the BuyLimit(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultOrder\(\)](#).

BuyStop

Places the pending order of Buy Stop type (buy at the price higher than current market price) with specified parameters.

```
bool BuyStop(
    double          volume,           // order volume
    double          price,           // order price
    const string    symbol=NULL,      // symbol
    double          sl=0.0,           // stop loss price
    double          tp=0.0,           // take profit price
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // order lifetime
    datetime        expiration=0,     // order expiration time
    const string    comment=""        // comment
)
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from [ENUM_ORDER_TYPE_TIME](#) enumeration.

expiration=0

[in] Order expiration time (used only if type_time=[ORDER_TIME_SPECIFIED](#)).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the BuyStop(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultOrder\(\)](#).

SellLimit

Places the pending order of Sell Limit type (sell at the price higher than current market price) with specified parameters.

```
bool SellLimit(
    double          volume,           // order volume
    double          price,           // order price
    const string    symbol=NULL,     // symbol
    double          sl=0.0,          // stop loss price
    double          tp=0.0,          // take profit price
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // order lifetime
    datetime        expiration=0,    // order expiration time
    const string    comment=""       // comment
)
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from [ENUM_ORDER_TYPE_TIME](#) enumeration.

expiration=0

[in] Order expiration time (used only if type_time=[ORDER_TIME_SPECIFIED](#)).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the SellLimit(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultOrder\(\)](#).

SellStop

Places the pending order of Sell Stop type (sell at the price lower than current market price) with specified parameters.

```
bool SellStop(
    double          volume,           // order volume
    double          price,           // order price
    const string    symbol=NULL,     // symbol
    double          sl=0.0,          // stop loss price
    double          tp=0.0,          // take profit price
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // order lifetime
    datetime        expiration=0,    // order expiration time
    const string    comment=""       // comment
)
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from [ENUM_ORDER_TYPE_TIME](#) enumeration.

expiration=0

[in] Order expiration time (used only if type_time=[ORDER_TIME_SPECIFIED](#)).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the SellStop(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server [return code](#)) using [ResultRetcode\(\)](#) and value returned by [ResultOrder\(\)](#).

Request

Gets the copy of the last request structure.

```
void Request (
    MqlTradeRequest& request // reference
) const
```

Parameters

request

[out] Reference to the copied structure.

Return Value

None.

RequestAction

Gets the trade operation type.

```
ENUM_TRADE_REQUEST_ACTIONS RequestAction() const
```

Return Value

Trade operation type used in the last request from [ENUM_TRADE_REQUEST_ACTIONS](#) enumeration.

RequestActionDescription

Gets the trade operation type as string.

```
string RequestActionDescription() const
```

Return Value

Trade operation type (as string) used in the last request.

RequestMagic

Gets the magic number of the Expert Advisor.

```
ulong RequestMagic() const
```

Return Value

The magic number (ID) of the Expert Advisor, used in the last request.

RequestOrder

Gets the order ticket.

```
ulong RequestOrder() const
```

Return Value

Order ticket of the last request.

RequestSymbol

Gets the name of the symbol.

```
string RequestSymbol() const
```

Return Value

The name of the symbol used in the last request.

RequestVolume

Gets the trade volume (in lots).

```
double RequestVolume() const
```

Return Value

The requested trade volume (in lots) used in the last request.

RequestPrice

Gets the order execution price.

```
double RequestPrice() const
```

Return Value

Order execution price used in the last request.

RequestStopLimit

Gets the price of pending order of Stop Limit type.

```
double RequestStoplimit() const
```

Return Value

The price of pending order of Stop Limit type used in the last request.

RequestSL

Gets the Stop Loss price of an order.

```
double RequestSL() const
```

Return Value

The Stop Loss price used in the last request.

RequestTP

Gets the Take Profit price of the order.

```
double RequestTP() const
```

Return Value

The Take Profit price used in the last request.

RequestDeviation

Gets the maximum price deviation from a requested price.

```
ulong RequestDeviation() const
```

Return Value

The maximum allowable price deviation from a requested price used in the last request.

RequestType

Gets the type of the order.

```
ENUM_ORDER_TYPE RequestType() const
```

Return Value

Order type used in the last request from [ENUM_ORDER_TYPE](#) enumeration.

RequestTypeDescription

Gets the type of the order (as string).

```
string RequestTypeDescription() const
```

Return Value

The order type (as string) used in the last request.

RequestTypeFilling

Gets the filling type of the order.

```
ENUM_ORDER_TYPE_FILLING RequestTypeFilling() const
```

Return Value

The filling type of the order from [ENUM_ORDER_TYPE_FILLING](#) used in the last request.

RequestTypeFillingDescription

Gets the filling type of the order (as string).

```
string RequestTypeFillingDescription() const
```

Return Value

The filling type (as string) of the order used in the last request.

RequestTypeTime

Gets the validity period of the order.

```
ENUM_ORDER_TYPE_TIME RequestTypeTime() const
```

Return Value

The validity period of the order from [ENUM_ORDER_TYPE_TIME](#) enumeration used in the last request.

RequestTypeTimeDescription

Gets the validity period of the order (as string).

```
string RequestTypeTimeDescription() const
```

Return Value

The validity period of the order (as string) used in the last request.

RequestExpiration

Gets the expiration time of a pending order.

```
datetime RequestExpiration() const
```

Return Value

The expiration time of a pending order used in the last request.

RequestComment

Gets the comment of the order.

```
string RequestComment() const
```

Return Value

The comment of the order used in the last request.

RequestPosition

Gets position ticket.

```
ulong RequestPosition() const
```

Return value

Position ticket used in the last request.

RequestPositionBy

Gets opposite position ticket.

```
ulong RequestPositionBy() const
```

Return value

Opposite position ticket used in the last request.

Result

Gets the copy of the structure of the last request result.

```
void Result(  
    MqlTradeResult& result    // reference  
    ) const
```

Parameters

result

[out] Reference to the structure of [MqlTradeResult](#) type for copying.

Return Value

None.

ResultRetcode

Gets the code of request execution result.

```
uint ResultRetcode() const
```

Return Value

The [code](#) of request result.

ResultRetcodeDescription

Gets the code of request execution result as a string.

```
string ResultRetcodeDescription() const
```

Return Value

[Code of the last request](#) result as a string.

ResultDeal

Gets the deal ticket.

```
ulong ResultDeal() const
```

Return Value

Deal ticket if the deal is executed.

ResultOrder

Gets the order ticket.

```
ulong ResultOrder() const
```

Return Value

Order ticket if the order is placed.

ResultVolume

Gets the volume of deal or order.

```
double ResultVolume() const
```

Return Value

Volume of deal or order confirmed by a broker.

ResultPrice

Gets the price confirmed by a broker.

```
double ResultPrice() const
```

Return Value

Price confirmed by a broker.

ResultBid

Gets the current bid price (the requote).

```
double ResultBid() const
```

Return Value

Current bid price (the requote).

ResultAsk

Gets the current ask price (the requote).

```
double ResultAsk() const
```

Return Value

Current ask price (the requote).

ResultComment

Gets the broker comment.

```
string ResultComment() const
```

Return Value

Broker comment to the operation.

CheckResult

Gets the copy of the structure of the last request check result.

```
void CheckResult(  
    MqlTradeCheckResult& check_result    // reference  
    ) const
```

Parameters

check_result

[out] Reference to the target structure of the [MqlTradeCheckResult](#) type for copying.

Return Value

None.

CheckResultRetcode

Gets the value of the retcode field of [MqlTradeCheckResult](#) type, filled while checking the request correctness.

```
uint CheckResultRetcode() const
```

Return Value

The value of the retcode field (error code) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultRetcodeDescription

Gets the string description of the retcode field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
string ResultRetcodeDescription() const
```

Return Value

The string description of the retcode field (Error code) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultBalance

Gets the value of the balance field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
double CheckResultBalance() const
```

Return Value

The value of the balance field (balance value that will be after the execution of the trade operation) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultEquity

Gets the value of the equity field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
double CheckResultEquity() const
```

Return Value

The value of the equity field (equity value that will be after the execution of the trade operation) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultProfit

Gets the value of a floating profit.

```
double CheckResultProfit() const
```

Return Value

The value of a floating profit after executing a trading operation.

CheckResultMargin

Gets the value of the margin field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
double CheckResultMargin() const
```

Return Value

The value of the margin field (margin required for the trade operation) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultMarginFree

Gets the value of the margin_free field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
double CheckResultMarginFree() const
```

Return Value

The value of the margin_free field (free margin that will be left after the execution of the trade operation) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultMarginLevel

Gets the value of the margin_level field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
double CheckResultMarginLevel() const
```

Return Value

The value of the margin_level field (margin level that will be set after the execution of the trade operation) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

CheckResultComment

The value of the comment field of [MqlTradeCheckResult](#) type filled while checking the request correctness.

```
string CheckResultComment() const
```

Return Value

The value of the comment field (comment to the reply code, error description) of [MqlTradeCheckResult](#) type filled while checking the request correctness.

PrintRequest

Prints the last request parameters into journal.

```
void PrintRequest() const
```

Return Value

None.

PrintResult

Prints the results of the last request into journal.

```
void PrintResult() const
```

Return Value

None.

FormatRequest

Prepares the formatted string with last request parameters.

```
string FormatRequest(  
    string&          str,          // string  
    const MqlTradeRequest& request // request  
) const
```

Parameters

str

[in] Target string passed by reference.

request

[in] A structure of [MqlTradeRequest](#) type with parameters of the last request.

Return Value

None.

FormatRequestResult

Prepares the formatted string with results of the last request execution.

```
string FormatRequestResult(  
    string&          str,          // string  
    const MqlTradeRequest& request, // request  
    const MqlTradeResult& result   // result  
) const
```

Parameters

str

[in] Target string passed by reference.

request

[in] A structure of [MqlTradeRequest](#) type with parameters of the last request.

result

[in] A structure of [MqlTradeResult](#) type with results of the last request.

Return Value

None.

CTerminalInfo

CTerminalInfo is a class for simplified access to the properties of mql5 program environment.

Description

CTerminalInfo class provides access to the properties of mql5 program environment.

Declaration

```
class CTerminalInfo : public CObject
```

Title

```
#include <Trade\TerminalInfo.mqh>
```

Inheritance hierarchy

CObject

CTerminalInfo

Class methods by groups

Methods for access to the properties of integer type	
<u>Build</u>	Gets the build number of the client terminal
<u>IsConnected</u>	Gets the information about connection to trade server
<u>IsDLLsAllowed</u>	Gets the information about permission of DLL usage
<u>IsTradeAllowed</u>	Gets the information about permission to trade
<u>IsEmailEnabled</u>	Gets the information about permission to send e-mails to SMTP server and login, specified in the terminal settings
<u>IsFtpEnabled</u>	Gets the information about permission to send trade reports to FTP server and login, specified in the terminal settings
<u>MaxBars</u>	Gets the information about maximum number of bars on chart
<u>CodePage</u>	Gets the information about the code page of the language in the client terminal
<u>CPUcores</u>	Gets the information about the CPU cores
<u>MemoryPhysical</u>	Gets the information about the physical memory (in Mb)

<u>MemoryTotal</u>	Gets the information about the total memory available for the terminal/agent process (in Mb)
<u>MemoryAvailable</u>	Gets the information about the free memory available for the terminal/agent process (in Mb)
<u>MemoryUsed</u>	Gets the information about the memory used by the terminal/agent process (in Mb)
<u>IsX64</u>	Gets the information about the type of the client terminal
<u>OpenCLSupport</u>	Gets the information about the version of OpenCL supported by video card
<u>DiskSpace</u>	Gets the information about free disk space (in Mb)
Methods for access to the properties of string type	
<u>Language</u>	Gets the language of the client terminal
<u>Name</u>	Gets the name of the client terminal
<u>Company</u>	Gets the company name of the client terminal
<u>Path</u>	Gets the folder of the client terminal
<u>DataPath</u>	Gets the data folder of the client terminal
<u>CommonDataPath</u>	Gets the common data folder of all client terminals, installed on the computer
Access to MQL5 API functions	
<u>InfoInteger</u>	Gets the value of the property of integer type
<u>InfoString</u>	Gets the value of property of string type

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

Build

Gets the build number of the client terminal.

```
int CBuild() const
```

Return Value

Build number of the client terminal.

Note

The terminal build number is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_BUILD](#) property).

IsConnected

Gets the information about connection to trade server.

```
bool IsConnected() const
```

Return Value

true - the terminal is connected to a trade server, otherwise - false.

Note

Connection status is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_CONNECTED](#) property).

IsDLLsAllowed

Gets the information about permission of DLL usage.

```
bool IsDLLsAllowed() const
```

Return Value

true - DLL usage is allowed, otherwise - false.

Note

Permission of DLL usage is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_DLLS_ALLOWED](#) property).

IsTradeAllowed

Gets the information about permission to trade.

```
bool IsTradeAllowed() const
```

Return Value

true - trade allowed, otherwise - false.

Note

Permission to trade is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_TRADE_ALLOWED](#) property).

IsEmailEnabled

Gets the information about permission to send e-mails to SMTP server and login specified in the terminal settings.

```
bool IsEmailEnabled() const
```

Return Value

true - sending e-mails is allowed, otherwise - false.

Note

Permission to send e-mails is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_EMAIL_ENABLED](#) property).

IsFtpEnabled

Gets the information about permission to send trade reports to FTP server and login specified in the terminal settings.

```
bool IsFtpEnabled() const
```

Return Value

true - sending trade reports to FTP server is allowed, otherwise - false.

Note

Permission to send trade reports is defined [TerminalInfoInteger\(\)](#) function ([TERMINAL_FTP_ENABLED](#) property).

MaxBars

Gets the maximum number of bars on chart specified in the terminal settings.

```
int MaxBars() const
```

Return Value

Maximum number of bars on the chart.

Note

The maximum number of bars on chart is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_MAXBARS](#) property).

CodePage

Gets the information about code page of the language in the terminal.

```
int CodePage () const
```

Return Value

Code page of the language in the client terminal.

Note

Code page of the language is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_CODEPAGE](#) property).

CPUCores

Gets the information about the number of CPU cores in the system.

```
int CPUCores () const
```

Return Value

Number of CPU cores in the system.

Note

The number of CPU cores is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_CPU_CORES](#) property).

MemoryPhysical

Gets the information about the physical memory (in Mb).

```
int MemoryPhysical() const
```

Return Value

Physical memory (in Mb).

Note

The physical memory is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_MEMORY_PHYSICAL](#) property).

MemoryTotal

Gets the information about the total memory available to the terminal/agent (in Mb).

```
int MemoryTotal() const
```

Return Value

Total memory (in Mb) available to the terminal/agent.

Note

The total memory available to the terminal/agent is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_MEMORY_TOTAL](#) property).

MemoryAvailable

Gets the information about the free memory available to the client terminal/agent (in Mb).

```
int MemoryTotal() const
```

Return Value

Free memory (in Mb) available to the terminal/agent.

Note

The free memory available to the client terminal/agent is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_MEMORY_TOTAL](#) property).

MemoryUsed

Gets the information about the memory used by the client terminal/agent (in Mb).

```
int MemoryUsed() const
```

Return Value

The memory used by the client terminal/agent (in Mb).

Note

The memory used by the client terminal/agent is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_MEMORY_USED](#) property).

IsX64

Gets the information about the type of the client terminal.

```
bool IsX64() const
```

Return Value

true - 64-bit version is used, otherwise - false.

Note

The type of the terminal is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_X64](#) property).

OpenCLSupport

Gets the information about the version of OpenCL supported by video card.

```
int OpenCLSupport() const
```

Return Value

OpenCL version having the following form: 0x00010002 = "1.2". 0 means that OpenCL is not supported.

Note

OpenCL version is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_OPENCL_SUPPORT](#) property).

DiskSpace

Gets the information about free disk space available for MQL5\Files folder of the terminal/agent (in Mb).

```
int MDiskSpace() const
```

Return Value

Free disk space.

Note

Free disk space is defined by [TerminalInfoInteger\(\)](#) function ([TERMINAL_DISK_SPACE](#) property).

Language

Gets the information about the language of the client terminal.

```
string Language() const
```

Return Value

Language used in the client terminal.

Note

The terminal language is defined by [TerminalInfoString\(\)](#) function ([TERMINAL_LANGUAGE](#) property).

Name

Gets the information on the name of the client terminal.

```
string Name() const
```

Return Value

Name of the client terminal.

Note

The name of the terminal is defined by [TerminalInfoString\(\)](#) function ([TERMINAL_NAME](#) property).

Company

Gets the information about the name of the company.

```
string Company() const
```

Return Value

The name of the company.

Note

The company name is defined by [TerminalInfoString\(\)](#) function ([TERMINAL_COMPANY](#) property).

Path

Gets the client terminal folder.

```
string Path() const
```

Return Value

The client terminal folder.

Note

The client terminal folder is used by [TerminalInfoString\(\)](#) function ([TERMINAL_PATH](#) property).

DataPath

Gets the information about the terminal data folder.

```
string DataPath() const
```

Return Value

Data folder of the client terminal.

Note

Client terminal data folder is defined by [TerminalInfoString\(\)](#) function ([TERMINAL_DATA_PATH](#) property).

CommonDataPath

Gets the common data folder of all client terminals installed on the computer.

```
string CommonDataPath() const
```

Return Value

Common data folder of all installed terminals.

Note

The common data folder of all installed terminals is defined by [TerminalInfoString\(\)](#) function ([COMMON_DATA_PATH](#) property).

InfoInteger

Returns the value of a corresponding integer property of the mql5 program environment.

```
int TerminalInfoInteger(  
    int property_id    // identifier of a property  
);
```

Parameters

property_id

[in] Identifier of a property. It can be one of the values of the enumeration [ENUM_TERMINAL_INFO_INTEGER](#).

Return Value

Value of int type.

Note

The property value is defined by [TerminalInfoInteger\(\)](#) function.

InfoString

The function returns the value of a corresponding property of the mql5 program environment. The property must be of string type.

```
string TerminalInfoString(  
    int property_id    // property ID  
);
```

Parameters

property_id

[in] Identifier of a property. It can be one the values of the enumeration [ENUM_TERMINAL_INFO_STRING](#).

Return Value

Value of string type.

Note

The property value is defined by [TerminalInfoString\(\)](#) function.

Classes for Creating and Testing Trading Strategies

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and especially testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert folder.

Base classes	Description
<u>CExpertBase</u>	Base class for all trading strategy classes
<u>CExpert</u>	Base class for Expert Advisor
<u>CExpertSignal</u>	Base class for Trading Signal classes
<u>CExpertTrailing</u>	Base class for Trailing Stop classes
<u>CExpertMoney</u>	Base class for Money Management classes

Trading signal classes	Description
<u>CSignalAC</u>	The module of signals based on market models of the indicator Accelerator Oscillator.
<u>CSignalAMA</u>	The module of signals based on market models of the indicator Adaptive Moving Average.
<u>CSignalAO</u>	The module of signals based on market models of the indicator Awesome Oscillator.
<u>CSignalBearsPower</u>	The module of signals based on market models of the oscillator Bears Power.
<u>CSignalBullsPower</u>	The module of signals based on market models of the oscillator Bulls Power.
<u>CSignalCCI</u>	The module of signals based on market models of the oscillator Commodity Channel Index.
<u>CSignalDeM</u>	The module of signals based on market models of the oscillator DeMarker.
<u>CSignalDEMA</u>	The module of signals based on market models of the indicator Double Exponential Moving Average.
<u>CSignalEnvelopes</u>	The module of signals based on market models of the indicator Envelopes.
<u>CSignalFrAMA</u>	The module of signals based on market models of the indicator Fractal Adaptive Moving Average.

Trading signal classes	Description
<u>CSignalITF</u>	The module of filtration of signals by time.
<u>CSignalMACD</u>	The module of signals based on market models of the oscillator MACD.
<u>CSignalMA</u>	The module of signals based on market models of the indicator Moving Average.
<u>CSignalSAR</u>	The module of signals based on market models of the indicator Parabolic SAR.
<u>CSignalRSI</u>	The module of signals based on market models of the oscillator Relative Strength Index.
<u>CSignalRVI</u>	The module of signals based on market models of the oscillator Relative Vigor Index.
<u>CSignalStoch</u>	The module of signals based on market models of the oscillator Stochastic.
<u>CSignalTRIX</u>	The module of signals based on market models of the oscillator Triple Exponential Average.
<u>CSignalTEMA</u>	The module of signals based on market models of the indicator Triple Exponential Moving Average.
<u>CSignalWPR</u>	The module of signals based on market models of the oscillator Williams Percent Range.

Trailing Stop classes	Description
<u>CTrailingFixedPips</u>	This class implements Trailing Stop algorithm based on fixed points
<u>CTrailingMA</u>	This class implements Trailing Stop algorithm based on the values of Moving Average indicator
<u>CTrailingNone</u>	A stub class, it does not use any Trailing Stop algorithm
<u>CTrailingPSAR</u>	This class implements Trailing Stop algorithm based on the values of Parabolic SAR indicator

Money Management classes	Description
<u>CMoneyFixedLot</u>	A class with an algorithm based on trading with predefined fixed lot size.
<u>CMoneyFixedMargin</u>	A class with an algorithm based on trading with predefined fixed margin.

<u>CMoneyFixedRisk</u>	A class with an algorithm based on trading with predefined risk.
<u>CMoneyNone</u>	A class with an algorithm based on trading with minimal allowed lot size.
<u>CMoneySizeOptimized</u>	A class with an algorithm based on trading with variable lot size, depending on the results of the previous deals.

Base classes for Expert Advisors

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and especially testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies classes) is placed in the terminal directory, in the Include\Expert folder.

Class	Description
<u>CExpertBase</u>	Base class for all trading strategy classes
<u>CExpert</u>	Base class for Expert Advisor
<u>CExpertSignal</u>	Base class for Trading Signal classes
<u>CExpertTrailing</u>	Base class for Trailing Stop classes
<u>CExpertMoney</u>	Base class for Money Management classes

CExpertBase

CExpertBase is a base class for the [CExpert](#) class and all auxiliary trading strategy classes.

Description

CExpertBase provides the data and methods, which are common to all objects of the Expert Advisor.

Declaration

```
class CExpertBase : public CObject
```

Title

```
#include <Expert\ExpertBase.mqh>
```

Inheritance hierarchy

[CObject](#)

CExpertBase

Direct descendants

[CExpert](#), [CExpertMoney](#), [CExpertSignal](#), [CExpertTrailing](#)

Class Methods by Groups

Public Methods:

Initialization	
virtual Init	Initializes the object
virtual ValidationSettings	Checks the settings
Parameters	
Symbol	Sets the symbol
Period	Sets the timeframe
Magic	Sets the Expert Advisor ID
Indicators and Timeseries	
virtual SetPriceSeries	Sets pointers to external timeseries (price series)
virtual SetOtherSeries	Sets pointers to external timeseries (non-price series)
virtual InitIndicators	Initializes the indicators and timeseries
Access to Protected Data	
InitPhase	Gets the current phase of object initialization

<u>TrendType</u>	Sets trend type
<u>UsedSeries</u>	Gets bitmask of timeseries used
<u>EveryTick</u>	Sets the "Every tick" flag
Access to Timeseries	
<u>Open</u>	Gets the element of the Open timeseries by index
<u>High</u>	Gets the element of the High timeseries by index
<u>Low</u>	Gets the element of the Low timeseries by index
<u>Close</u>	Gets the element of the Close timeseries by index
<u>Spread</u>	Gets the element of the Spread timeseries by index
<u>Time</u>	Gets the element of the Time timeseries by index
<u>TickVolume</u>	Gets the element of the TickVolume timeseries by index
<u>RealVolume</u>	Gets the element of the RealVolume timeseries by index

Protected Methods:

Initialization of Timeseries	
<u>InitOpen</u>	Open timeseries initialization method
<u>InitHigh</u>	High timeseries initialization method
<u>InitLow</u>	Low timeseries initialization method
<u>InitClose</u>	Close timeseries initialization method
<u>InitSpread</u>	Spread timeseries initialization method
<u>InitTime</u>	Time timeseries initialization method
<u>InitTickVolume</u>	TickVolume timeseries initialization method
<u>InitRealVolume</u>	RealVolume timeseries initialization method
Service Methods	
virtual <u>PriceLevelUnit</u>	Gets the price level unit
virtual <u>StartIndex</u>	Gets the index of starting bar to analyze

virtual CompareMagic	Compares the Expert Advisor ID with the specified value
--------------------------------------	---

Methods inherited from class CObjectPrev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

InitPhase

Gets the current phase of the object initialization.

```
ENUM_INIT_PHASE InitPhase()
```

Return Value

Current phase of the object initialization.

Note

The object initialization consist of several phases:

1. Start initialization.

- start - after finish of the constructor
- finish - after successful completion of the [Init\(...\)](#) method.
- allowed - call of the [Init\(...\)](#) method
- not allowed - call of the [ValidationSettings\(\)](#) method and other initialization methods

2. Parameters setting phase. In this phase you need to set all the object parameters, used for creation of indicators.

- start - after successful completion of the [Init\(...\)](#) method
- finish - after successful completion of the [ValidationSettings\(\)](#) method
- allowed - call of [Symbol\(...\)](#) and [Period\(...\)](#) methods
- not allowed - call of the [Init\(...\)](#), [SetPriceSeries\(...\)](#), [SetOtherSeries\(...\)](#) and [InitIndicators\(...\)](#) methods

3. Checking of parameters.

- start - after successful completion of the [ValidationSettings\(\)](#) method
- finish - after successful completion of the [InitIndicators\(...\)](#) method
- allowed - call of the [Symbol\(...\)](#), [Period\(...\)](#) and [InitIndicators\(...\)](#) methods
- not allowed - call of any other initialization methods

4. Finish of initialization.

- start - after successful completion of the [InitIndicators\(...\)](#) method
- not allowed - call of initialization methods

TrendType

Sets trend type.

```
void TrendType(  
    M_TYPE_TREND value // value  
)
```

Parameters

value

[in] New value of trend type.

Return Value

None.

UsedSeries

Gets the bitmask of timeseries used.

```
int UsedSeries()
```

Return Value

The list of used timeseries as bitmask.

Note

If the bit is set, the corresponding timeseries is used, if it is not set, the timeseries is not used.

The bit-timeseries correspondence:

- bit 0 - Open timeseries,
- bit 1 - High timeseries,
- bit 2 - Low timeseries,
- bit 3 - Close timeseries,
- bit 4 - Spread timeseries,
- bit 5 - Time timeseries,
- bit 6 - TickVolume timeseries,
- bit 7 - RealVolume timeseries.

EveryTick

Sets the "Every tick" flag.

```
void EveryTick(  
    bool    value        // flag  
)
```

Parameters

value

[in] New value of a flag.

Return Value

None.

Note

If the flag is set, each price (tick) change at a working symbol is processed.

If the flag is not set, the processing method is called only at a new bar on the working timeframe and symbol.

Open

Gets the element of the Open timeseries by index.

```
double Open(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Open timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

High

Gets the element of the High timeseries by index.

```
double High(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the High timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Low

Gets the element of the Low timeseries by index.

```
double Low(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Low timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Close

Gets the element of the Close timeseries by index.

```
double Close (  
    int ind      // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Close timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Spread

Gets the element of the Spread timeseries by index.

```
double Spread(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Spread timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Time

Gets the element of the Time timeseries by index.

```
datetime Time(  
    int ind      // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Time timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

TickVolume

Gets the element of the TickVolume timeseries by index.

```
long TickVolume(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the TickVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

RealVolume

Gets the element of the RealVolume timeseries by index.

```
long RealVolume(  
    int ind // index  
)
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the RealVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Init

Initializes the object.

```
bool Init(  
    CSymbolInfo    symbol,    // symbol  
    ENUM_TIMEFRAMES period,    // timeframe  
    double         point      // point  
)
```

Parameters

symbol

[in] Pointer to the object of [CSymbolInfo](#) type for access to symbol information.

period

[in] Timeframe ([ENUM_TIMEFRAMES](#) enumeration).

point

[in] The "weight" of 2/4-digit point.

Return Value

true - successful completion, otherwise - false.

Symbol

Sets the object symbol.

```
bool Symbol(  
    string name // symbol  
)
```

Parameters

name

[in] Symbol.

Return Value

true - successful, otherwise - false.

Note

The setting of working symbol is necessary if the object uses the symbol different from symbol defined at the first initialization.

Period

Sets the object timeframe.

```
bool Period(  
    ENUM_TIMEFRAMES value    // timeframe  
)
```

Parameters

value

[in] Timeframe.

Return Value

true - successful, otherwise - false.

Note

The setting of working timeframe is necessary if the object uses the timeframe different from timeframe defined at the initialization.

Magic

Sets the Expert Advisor ID.

```
void Magic(  
    ulong value    // magic  
)
```

Parameters

value

[in] Expert Advisor ID.

Return Value

None.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

SetPriceSeries

Sets pointers to external price series.

```
virtual bool SetPriceSeries(  
    CiOpen*   open,      // pointer  
    CiHigh*   high,      // pointer  
    CiLow*    low,       // pointer  
    CiClose*  close      // pointer  
)
```

Parameters

open

[in] Pointer to Open timeseries.

high

[in] Pointer to High timeseries.

low

[in] Pointer to Low timeseries.

close

[in] Pointer to Close timeseries.

Return Value

true - successful, otherwise - false.

Note

The setting of pointers to external timeseries (price series) is necessary if the object uses the symbol and timeframe (set during the first initialization) and price timeseries necessary for further work.

SetOtherSeries

Sets pointers to external non-price series.

```
virtual bool SetOtherSeries(  
    CiSpread*      spread,      // pointer  
    CiTime*        time,        // pointer  
    CiTickVolume*  tick_volume, // pointer  
    CiRealVolume*  real_volume  // pointer  
)
```

Parameters

spread

[in] Pointer to Spread timeseries.

time

[in] Pointer to Time timeseries.

tick_volume

[in] Pointer to TickVolume timeseries.

real_volume

[in] Pointer to RealVolume timeseries.

Return Value

true - successful, otherwise - false.

Note

The setting of pointers to external timeseries (price series) is necessary if the object uses the symbol and timeframe (set during the first initialization) and price timeseries necessary for further work.

InitIndicators

Initializes all indicators and time series.

```
virtual bool InitIndicators(  
    CIndicators* indicators=NULL    // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The timeseries are initialized only if the object uses the symbol or timeframe different from the symbol or timeframe defined at initialization.

InitOpen

Initializes the Open timeseries.

```
bool InitOpen(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Open timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitHigh

Initializes the High timeseries.

```
bool InitHigh(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The High timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitLow

Initializes the Low timeseries.

```
bool InitLow(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Low timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitClose

Initializes the Close timeseries.

```
bool InitClose(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Close timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitSpread

Initializes the Spread timeseries.

```
bool InitSpread(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Spread timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitTime

Initializes the Time timeseries.

```
bool InitTime(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Time timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitTickVolume

Initializes the TickVolume timeseries.

```
bool InitTickVolume(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The TickVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitRealVolume

Initializes the RealVolume timeseries.

```
bool InitRealVolume(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The RealVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

PriceLevelUnit

Gets the price level unit.

```
virtual double PriceLevelUnit ()
```

Return Value

The value of price level unit.

Note

The method of a base class returns the "weight" of the 2/4 digits point.

StartIndex

Gets the index of starting bar to analyze.

```
virtual int StartIndex()
```

Return Value

The index of starting bar to analyze.

Note

The method returns 0 if the flag to analyze current bar is set to true (analysis from the current bar). If the flag is not set, it returns 1 (analysis from the last completed bar).

CompareMagic

Compares the Expert Advisor ID (magic) with the specified value.

```
virtual bool CompareMagic(  
    ulong magic    // identifier  
)
```

Parameters

magic

[in] Identifier value to compare.

Return Value

true - identifier matches the specified one, otherwise - false.

CExpert

CExpert is a base class for trading strategies.

It already has some elementary trading "skills". It has built-in algorithms for working with time series and indicators and a set of virtual methods for trading strategy.

How to use it:

1. Prepare an algorithm of the strategy;
2. Create your own class, inherited from CExpert class;
3. Override the virtual methods in your class with your own algorithms.

Description

The CExpert class is a set of virtual methods for implementation of trading strategies.

Note

A position is recognized as belonging to an Expert Advisor and managed by it based on the pair of properties `m_symbol` and `m_magic`. In the "hedging" mode, multiple positions can be opened for the same symbol, therefore the `m_magic` value is important.

Declaration

```
class CExpert : public CExpertBase
```

Title

```
#include <Expert\Expert.mqh>
```

Inheritance hierarchy

CObject

CExpertBase

CExpert

Class Methods by Groups

Initialization	
<u>Init</u>	Class instance initialization method
virtual <u>InitSignal</u>	Initializes Trading Signal object
virtual <u>InitTrailing</u>	Initializes Trailing Stop object
virtual <u>InitMoney</u>	Initializes Money Management object
virtual <u>InitTrade</u>	Initializes Trade object
virtual <u>ValidationSettings</u>	Checks the settings
virtual <u>InitIndicators</u>	Initializes indicators and timeseries

virtual InitParameters	Expert Advisor parameters initialization method
virtual Deinit	Class instance deinitialization method
virtual DeinitSignal	Deinitializes Trading Signal object
virtual DeinitTrailing	Deinitializes Trailing Stop object
virtual DeinitMoney	Deinitializes Money Management object
virtual DeinitTrade	Deinitializes Trade object
virtual DeinitIndicators	Deinitializes indicators and timeseries
Parameters	
Magic	Sets the Expert Advisor ID
MaxOrders	Gets/sets the maximum amount of allowed orders
OnTickProcess	Sets a flag to proceed the "OnTick" event
OnTradeProcess	Sets a flag to proceed the "OnTrade" event
OnTimerProcess	Sets a flag to proceed the "OnTimer" event
OnChartEventProcess	Sets a flag to proceed the "OnChartEvent" event
OnBookEventProcess	Sets a flag to proceed the "OnBookEvent" event
Event Processing Methods	
OnTick	OnTick event handler
OnTrade	OnTrade event handler
OnTimer	OnTimer event handler
OnChartEvent	OnChartEvent event handler
OnBookEvent	OnBookEvent event handler
Update Methods	
Refresh	Updates all data
Processing	
Processing	Main processing algorithm
Market Entry Methods	
CheckOpen	Checks position opening conditions
CheckOpenLong	Checks conditions to open long position
CheckOpenShort	Checks conditions to open short position
OpenLong	Opens a long position
OpenShort	Opens a short position

Market Exit Methods	
<u>CheckClose</u>	Checks conditions to close current position
<u>CheckCloseLong</u>	Checks conditions to close long position
<u>CheckCloseShort</u>	Checks conditions to close short position
<u>CloseAll</u>	Closes the opened position and deletes all orders
<u>Close</u>	Closes the opened position
<u>CloseLong</u>	Closes the long position
<u>CloseShort</u>	Closes the short position
Position Reverse Methods	
<u>CheckReverse</u>	Checks conditions to reverse opened position
<u>CheckReverseLong</u>	Checks conditions to reverse long position
<u>CheckReverseShort</u>	Checks conditions to reverse short position
<u>ReverseLong</u>	Performs reverse operation of long position
<u>ReverseShort</u>	Performs reverse operation of short position
Position/Order Trailing Methods	
<u>CheckTrailingStop</u>	Checks conditions to modify position parameters
<u>CheckTrailingStopLong</u>	Checks Trailing Stop conditions of long position
<u>CheckTrailingStopShort</u>	Checks Trailing Stop conditions of short position
<u>TrailingStopLong</u>	Performs Trailing Stop for long position
<u>TrailingStopShort</u>	Performs Trailing Stop for short position
<u>CheckTrailingOrderLong</u>	Checks Trailing Stop conditions of Buy Limit/Stop order
<u>CheckTrailingOrderShort</u>	Checks Trailing Stop conditions of Sell Limit/Stop order
<u>TrailingOrderLong</u>	Performs Trailing Stop for Buy Limit/Stop order
<u>TrailingOrderShort</u>	Performs Trailing Stop for Sell Limit/Stop order
Order Delete Methods	
<u>CheckDeleteOrderLong</u>	Checks conditions to delete Buy order
<u>CheckDeleteOrderShort</u>	Checks conditions to delete Sell order
<u>DeleteOrders</u>	Deletes all orders

<u>DeleteOrder</u>	Deletes Stop/Limit order
<u>DeleteOrderLong</u>	Deletes Buy Limit/Stop order
<u>DeleteOrderShort</u>	Deletes Sell Limit/Stop order
Trade Volume Methods	
<u>LotOpenLong</u>	Gets trade volume for buy operation
<u>LotOpenShort</u>	Gets trade volume for sell operation
<u>LotReverse</u>	Gets trade volume for position reverse operation
Trade History Methods	
<u>PrepareHistoryDate</u>	Sets starting date for trade history tracking
<u>HistoryPoint</u>	Creates a checkpoint of trade history (saves number of positions, orders, deals and historical orders)
<u>CheckTradeState</u>	Compares the current state with the saved one and calls the corresponding event handler
Event flags	
<u>WaitEvent</u>	Sets the trading event waiting flag
<u>NoWaitEvent</u>	Resets the trading event waiting flag
Trade Event Processing Methods	
<u>TradeEventPositionStopTake</u>	Event handler of the "Position Stop Loss/Take Profit triggered" event
<u>TradeEventOrderTriggered</u>	Event handler of the "Pending Order Triggered" event
<u>TradeEventPositionOpened</u>	Event handler of the "Position Opened" event
<u>TradeEventPositionVolumeChanged</u>	Event handler of the "Position Volume Changed" event
<u>TradeEventPositionModified</u>	Event handler of the "Position Modified" event
<u>TradeEventPositionClosed</u>	Event handler of the "Position Closed" event
<u>TradeEventOrderPlaced</u>	Event handler of the "Pending Order Placed" event
<u>TradeEventOrderModified</u>	Event handler of the "Pending Order Modified" event
<u>TradeEventOrderDeleted</u>	Event handler of the "Pending Order Deleted" event
<u>TradeEventNotIdentified</u>	Event handler of the non-identified event

Service methods	
TimeframeAdd	Adds a timeframe to track
TimeframesFlags	Forms timeframe flags
SelectPosition	Selects a position to work with

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

Init

Class instance initialization method.

```
bool Init(  
    string          symbol,          // symbol  
    ENUM_TIMEFRAMES period,          // timeframe  
    bool            every_tick,      // flag  
    ulong           magic            // Expert Advisor identifier  
)
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe from [ENUM_TIMEFRAMES](#) enumeration.

every_tick

[in] Flag.

magic

[in] Expert Advisor ID (Magic number).

Return Value

None.

Note

If *every_tick* is set to true, the [Processing\(\)](#) method is called at each tick of the working symbol. otherwise, the [Processing\(\)](#) is called only when a new bar is formed on the working timeframe of the EA's working symbol.

Magic

Sets the Expert Advisor ID (magic).

```
void Magic(  
    ulong value    // new value  
)
```

Parameters

value

[in] New value of Expert Advisor ID.

Return Value

None.

Note

If EA's identifier is changed, the same identifier is assigned to all auxiliary objects.

Implementation

```
//+-----+  
//| Sets magic number for object and its dependent objects |  
//| INPUT:  value - new value of magic number.           |  
//| OUTPUT: no.                                           |  
//| REMARK: no.                                           |  
//+-----+  
void CExpert::Magic(ulong value)  
{  
    if(m_trade!=NULL)    m_trade.SetExpertMagicNumber(value);  
    if(m_signal!=NULL)   m_signal.Magic(value);  
    if(m_money!=NULL)    m_money.Magic(value);  
    if(m_trailing!=NULL) m_trailing.Magic(value);  
    /---  
    CExpertBase::Magic(value);  
}
```


InitSignal

Initializes trading signal object.

```
virtual bool InitSignal(  
    CExpertSignal*    signal=NULL,        // pointer  
)
```

Parameters

signal

[in] Pointer to [CExpertSignal](#) class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If signal is NULL, the [CExpertSignal](#) class will be used for initialization (it does nothing).

InitTrailing

Initializes trailing stop object.

```
virtual bool InitTrailing(  
    CExpertTrailing*   trailing=NULL,           // pointer  
)
```

Parameters

trailing

[in] Pointer to [CExpertTrailing](#) class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If trailing is NULL, the [ExpertTrailing](#) class will be used for initialization (it does nothing).

InitMoney

Initializes the money management object.

```
virtual bool InitMoney(  
    CExpertMoney* money=NULL, // pointer  
)
```

Parameters

money

[in] Pointer to [CExpertMoney](#) class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If money is NULL, the [CExpertMoney](#) class will be used for initialization (it uses the minimum lot).

InitTrade

Initializes the trade object.

```
virtual bool InitTrade(  
    ulong          magic,          // identifier  
    CExpertTrade*  trade=NULL     // pointer  
)
```

Parameters

magic

[in] Expert Advisor ID (will be used in trade requests).

trade

[in] Pointer to trade object.

Return Value

true - successful, otherwise - false.

Deinit

Class instance deinitialization method.

```
virtual void Deinit()
```

Return Value

None.

OnTickProcess

Sets the [OnTick](#) event handling flag.

```
void OnTickOProcess(  
    bool    value    // flag  
)
```

Parameters

value

[in] [OnTick](#) event handling flag.

Return Value

None.

Note

If the flag is true, the [OnTick](#) event is handled. By default, the flag is set to true.

OnTradeProcess

Sets the [OnTrade](#) event handling flag.

```
void OnTradeProcess(  
    bool    value    // flag  
)
```

Parameters

value

[in] [OnTrade](#) event handling flag.

Return Value

None.

Note

If the flag is true, the [OnTrade](#) event is handled. By default, the flag is set to false.

OnTimerProcess

Sets the [OnTimer](#) event handling flag.

```
void OnTimerProcess(  
    bool    value    // flag  
)
```

Parameters

value

[in] [OnTimer](#) event handling flag.

Return Value

None.

Note

If the flag is true, the [OnTimer](#) event is handled. By default, the flag is set to false.

OnChartEventProcess

Sets a flag to handle the [OnChartEvent](#) event.

```
void OnChartEventProcess (
    bool    value    // flag
)
```

Parameters

value

[in] Flag to handle the [OnChartEvent](#) event.

Return Value

None.

Note

If the flag is true, the [OnChartEvent](#) event is handled. By default, the flag is set to false.

OnBookEventProcess

Sets a flag to handle the [OnBookEvent](#) event.

```
void OnChartEventProcess (
    bool    value    // flag
)
```

Parameters

value

[in] Flag to handle the [OnBookEvent](#) event.

Return Value

None.

Note

If the flag is true, the [OnBookEvent](#) event is handled. By default, the flag is set to false.

MaxOrders (Get Method)

Gets the maximum amount of allowed orders.

```
int MaxOrders()
```

Return Value

Maximum amount of allowed orders.

MaxOrders (Set Method)

Sets the maximum amount of allowed orders.

```
void MaxOrders(  
    int    max_orders    // amount of orders  
)
```

Parameters

max_orders

[in] New value of maximum amount of allowed orders.

Return Value

None.

Note

By default, the maximum amount of allowed orders is equal to 1.

Signal

Gets the pointer to the trade signal object.

```
CExpertSignal* Signal() const
```

Return Value

Pointer to the trade signal object.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

It also checks the settings of all the Expert Advisor objects.

InitIndicators

Initializes necessary indicators and timeseries.

```
virtual bool InitIndicators(  
    CIndicators* indicators=NULL    // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful completion, otherwise - false.

Note

The timeseries are initialized if the object uses a symbol or timeframe other than the one defined in the initialization.

Indicators and timeseries of all auxiliary EA objects are initialized.

OnTick

[OnTick](#) event handler.

```
virtual void OnTick()
```

Return Value

None.

OnTrade

[OnTrade](#) event handler.

```
virtual void OnTrade()
```

Return Value

None.

OnTimer

[OnTimer](#) event handler.

```
virtual void OnTimer ()
```

Return Value

None.

OnChartEvent

OnChartEvent event handler.

```
virtual void OnChartEvent(  
    const int      id,          // event id  
    const long&    lparam,      // long type event parameter  
    const double   dparam,      // double type event parameter  
    const string   sparam       // string type event parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type.

dparam

[in] Event parameter of double type.

sparam

[in] Event parameter of string type.

Return Value

None.

OnBookEvent

[OnBookEvent](#) event handler.

```
virtual void OnBookEvent(  
    const string&    symbol        // symbol  
)
```

Parameters

symbol

[in] Symbol of [OnBookEvent](#) event.

Return Value

None.

InitParameters

Initializes parameters of Expert Advisor.

```
virtual bool InitParameters()
```

Return Value

true - successful, otherwise - false.

Note

The InitParameters() function of [CExpert](#) base class does nothing and always returns true.

DeinitTrade

Deinitializes trade object.

```
virtual void DeinitTrade()
```

Return Value

None.

DeinitSignal

Deinitializes trade signal object.

```
virtual void DeinitSignal()
```

Return Value

None.

DeinitTrailing

Deinitializes trailing object.

```
virtual void DeinitTrailing()
```

Return Value

None.

DeinitMoney

Deinitializes money management object.

```
virtual void DeinitMoney()
```

Return Value

None.

DeinitIndicators

Deinitializes indicators and timeseries.

```
virtual void DeinitIndicators ()
```

Return Value

None.

Note

Indicators and timeseries of all auxiliary EA objects are deinitialized.

Refresh

Updates all necessary data.

```
virtual bool Refresh()
```

Return Value

true - further tick processing is needed, otherwise - false.

Note

It allows to determine the need of tick processing. If it is needed, it updates all quotes and timeseries and indicators data and returns true.

Implementation

```
//+-----+
//| Refreshing data for processing |
//| INPUT: no. |
//| OUTPUT: true-if successful, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::Refresh()
{
    MqlDateTime time;
    //-- refresh rates
    if(!m_symbol.RefreshRates()) return(false);
    //-- check need processing
    TimeToStruct(m_symbol.Time(),time);
    if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
        if((m_period_flags & TimeframesFlags(time))==0) return(false);
    m_last_tick_time=time;
    //-- refresh indicators
    m_indicators.Refresh();
    //-- ok
    return(true);
}
```


Processing

Main processing algorithm.

```
virtual bool Processing()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It does the following steps:

1. Checks the presence of the opened position on the symbol. If there is no opened position, skip steps 2, 3, and 4.
2. Checks conditions to reverse opened position ([CheckReverse\(\)](#) method). If position has been "reversed", exit.
3. Checks conditions to close position ([CheckClose\(\)](#) method). If position has been closed, skip step 4.
4. Checks conditions to modify position parameters ([CheckTrailingStop\(\)](#) method). If position parameters have been modified, exit.
5. Check the presence of pending orders on the symbol. If there is no any pending order, go to step 9.
6. Checks condition to delete order ([CheckDeleteOrderLong\(\)](#) for buy pending orders or [CheckDeleteOrderShort\(\)](#) for sell pending orders). If the order has been deleted, go to step 9.
7. Check conditions to modify pending order parameters ([CheckTrailingOrderLong\(\)](#) for buy orders or [CheckTrailingOrderShort\(\)](#) for sell orders). If the order parameters have been modified, exit.
8. Exit.
9. Checks conditions to open position ([CheckOpen\(\)](#) method).

If you want to implement your own algorithm, you need to override the [Processing\(\)](#) method of the descendant class.

Implementation


```

//+-----+
//| Main function                                     |
//| INPUT:  no.                                       |
//| OUTPUT: true-if any trade operation processed, false otherwise. |
//| REMARK: no.                                       |
//+-----+
bool CExpert::Processing()
{
//--- check if open positions
    if(m_position.Select(m_symbol.Name()))
    {
        //--- open position is available
        //--- check the possibility of reverse the position
        if(CheckReverse()) return(true);
        //--- check the possibility of closing the position/delete pending orders
        if(!CheckClose())
        {
            //--- check the possibility of modifying the position
            if(CheckTrailingStop()) return(true);
            //--- return without operations
            return(false);
        }
    }
//--- check if placed pending orders
    int total=OrdersTotal();
    if(total!=0)
    {
        for(int i=total-1;i>=0;i--)
        {
            m_order.SelectByIndex(i);
            if(m_order.Symbol()!=m_symbol.Name()) continue;
            if(m_order.OrderType()==ORDER_TYPE_BUY_LIMIT || m_order.OrderType()==ORDER_T
            {
                //--- check the ability to delete a pending order to buy
                if(CheckDeleteOrderLong()) return(true);
                //--- check the possibility of modifying a pending order to buy
                if(CheckTrailingOrderLong()) return(true);
            }
            else
            {
                //--- check the ability to delete a pending order to sell
                if(CheckDeleteOrderShort()) return(true);
                //--- check the possibility of modifying a pending order to sell
                if(CheckTrailingOrderShort()) return(true);
            }
            //--- return without operations
            return(false);
        }
    }
//--- check the possibility of opening a position/setting pending order
    if(CheckOpen()) return(true);
//--- return without operations
    return(false);
}

```


SelectPosition

Selects a position to work with.

```
void SelectPosition()
```

Return Value

No.

Implementation

```
//+-----+
//| Position select                                     |
//| INPUT:  no.                                         |
//| OUTPUT: no.                                         |
//| REMARK: no.                                         |
//+-----+
bool CExpert::SelectPosition(void)
{
    bool res=false;
//---
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        res=m_position.SelectByMagic(m_symbol.Name(),m_magic);
    else
        res=m_position.Select(m_symbol.Name());
//---
    return(res);
}
```


CheckOpen

Checks conditions to open a position.

```
virtual bool CheckOpen()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open long ([CheckOpenLong\(\)](#)) and short ([CheckOpenShort\(\)](#)) positions.

Implementation

```
//+-----+
//| Check for position open or limit/stop order set |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckOpen()
{
    if(CheckOpenLong()) return(true);
    if(CheckOpenShort()) return(true);
    //--- return without operations
    return(false);
}
```


CheckOpenLong

Checks necessity and conditions to open long position.

```
virtual bool CheckOpenLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open a long position (CheckOpenLong() method of Signal object) and does that with the parameters set by Signal object ([OpenLong\(\)](#) method) if the conditions are met.

Implementation

```
//+-----+
//| Check for long position open or limit/stop order set |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckOpenLong()
{
    double price=EMPTY_VALUE;
    double sl=0.0;
    double tp=0.0;
    datetime expiration=TimeCurrent();
    //-- check signal for long enter operations
    if(m_signal.CheckOpenLong(price,sl,tp,expiration))
    {
        if(!m_trade.SetOrderExpiration(expiration))
        {
            m_expiration=expiration;
        }
        return(OpenLong(price,sl,tp));
    }
    //-- return without operations
    return(false);
}
```


CheckOpenShort

Checks necessity and conditions to open a short position.

```
virtual bool CheckOpenShort()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open a short position (CheckOpenShort() method of Signal object) and does that with the parameters set by Signal object ([OpenShort\(\)](#) method) if the conditions are met.

Implementation

```
//+-----+
//| Check for short position open or limit/stop order set |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckOpenShort()
{
    double price=EMPTY_VALUE;
    double sl=0.0;
    double tp=0.0;
    datetime expiration=TimeCurrent();
//--- check signal for short enter operations
    if(m_signal.CheckOpenShort(price,sl,tp,expiration))
    {
        if(!m_trade.SetOrderExpiration(expiration))
        {
            m_expiration=expiration;
        }
        return(OpenShort(price,sl,tp));
    }
//--- return without operations
    return(false);
}
```


OpenLong

Opens a long position.

```
virtual bool OpenLong(  
    double price,    // price  
    double sl,       // Stop Loss  
    double tp        // Take Profit  
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets trading volume ([LotOpenLong\(...\)](#) method) and opens a long position (Buy() method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+-----+  
//| Long position open or limit/stop order set |  
//| INPUT:  price - price,                    |  
//|         sl    - stop loss,                |  
//|         tp    - take profit.              |  
//| OUTPUT: true-if trade operation processed, false otherwise. |  
//| REMARK: no.                              |  
//+-----+  
bool CExpert::OpenLong(double price,double sl,double tp)  
{  
    if(price==EMPTY_VALUE) return(false);  
    //--- get lot for open  
    double lot=LotOpenLong(price,sl);  
    //--- check lot for open  
    if(lot==0.0) return(false);  
    //---  
    return(m_trade.Buy(lot,price,sl,tp));  
}
```


OpenShort

Opens a short position.

```
virtual bool OpenShort(
    double price, // price
    double sl,    // Stop Loss
    double tp     // Take Profit
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets trading volume ([LotOpenShort\(\)](#) method) and opens a short position (by calling Sell method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+-----+
//| Short position open or limit/stop order set |
//| INPUT:  price - price,                      |
//|         sl    - stop loss,                  |
//|         tp    - take profit.                |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no.                                |
//+-----+
bool CExpert::OpenShort(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE) return(false);
//--- get lot for open
    double lot=LotOpenShort(price,sl);
//--- check lot for open
    if(lot==0.0) return(false);
//---
    return(m_trade.Sell(lot,price,sl,tp));
}
```


CheckReverse

Checks necessity and conditions to reverse an open position.

```
virtual bool CheckReverse()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse long ([CheckReverseLong\(\)](#)) and short ([CheckReverseShort\(\)](#)) positions.

Implementation

```
//+-----+
//| Check for position reverse |
//| INPUT:  no.                |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                |
//+-----+
bool CExpert::CheckReverse()
{
    if(m_position.PositionType()==POSITION_TYPE_BUY)
    {
        //--- check the possibility of reverse the long position
        if(CheckReverseLong()) return(true);
    }
    else
        //--- check the possibility of reverse the short position
        if(CheckReverseShort()) return(true);
    //--- return without operations
    return(false);
}
```


CheckReverseLong

Checks necessity and conditions to reverse a long position.

```
virtual bool CheckReverseLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse a long position (CheckReverseLong() method of Signal object) and perform reverse operation of the current long position with the parameters set by Signal object ([ReverseLong\(...\)](#) method) if the conditions are met.

Implementation

```
//+-----+
//| Check for long position reverse |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckReverseLong()
{
    double price=EMPTY_VALUE;
    double sl=0.0;
    double tp=0.0;
    datetime expiration=TimeCurrent();
    //--- check signal for long reverse operations
    if(m_signal.CheckReverseLong(price,sl,tp,expiration)) return(ReverseLong(price,sl,tp,expiration));
    //--- return without operations
    return(false);
}
```


CheckReverseShort

Checks necessity and conditions to reverse a short position.

```
virtual bool CheckReverseLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse a short position (CheckReverseShort() method of Signal object) and perform reverse operation of the current short position with the parameters set by Signal object ([ReverseShort\(\)](#) method) if the conditions are met.

Implementation

```
//+-----+
//| Check for short position reverse |
//| INPUT:  no.                      |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                      |
//+-----+
bool CExpert::CheckReverseShort()
{
    double price=EMPTY_VALUE;
    double sl=0.0;
    double tp=0.0;
    datetime expiration=TimeCurrent();
    //-- check signal for short reverse operations
    if(m_signal.CheckReverseShort(price,sl,tp,expiration)) return(ReverseShort(price,sl,expiration));
    //-- return without operations
    return(false);
}
```


ReverseLong

Performs reverse operation of a long position.

```
virtual bool  ReverseLong(  
    double    price,      // price  
    double    sl,         // Stop Loss  
    double    tp          // Take Profit  
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets the position reverse volume ([LotReverse\(\)](#) method) and reverses a long position (Sell() method of Trade object) if trading volume is not equal to 0.

In the "hedging" mode of position accounting, position reversal is performed as the closure of the existing position and opening of a new opposite one with the remaining volume.

Implementation


```
//+-----+
//| Long position reverse |
//| INPUT:  price - price, |
//|         sl    - stop loss, |
//|         tp    - take profit. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::ReverseLong(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE)
        return(false);
//--- get lot for reverse
    double lot=LotReverse(sl);
//--- check lot
    if(lot==0.0)
        return(false);
//---
    bool result=true;
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
    {
        //--- first close existing position
        lot-=m_position.Volume();
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    }
    if(result)
        result=m_trade.Sell(lot,price,sl,tp);
//---
    return(result);
}
```


ReverseShort

Performs reverse operation of a short position.

```
virtual bool ReverseShort(  
    double price, // price  
    double sl,    // Stop Loss  
    double tp     // Take Profit  
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets position reverse volume ([LotReverse\(...\)](#) method) and perform trade operation of the short position reverse (Buy() method of Trade object) if trading volume is not equal to 0.

In the "hedging" mode of position accounting, position reversal is performed as the closure of the existing position and opening of a new opposite one with the remaining volume.

Implementation


```
//+-----+
//| Short position reverse                               |
//| INPUT:  price - price,                               |
//|         sl    - stop loss,                           |
//|         tp    - take profit.                         |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                                           |
//+-----+
bool CExpert::ReverseLong(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE)
        return(false);
//--- get lot for reverse
    double lot=LotReverse(sl);
//--- check lot
    if(lot==0.0)
        return(false);
//---
    bool result=true;
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
    {
        //--- first close existing position
        lot-=m_position.Volume();
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    }
    if(result)
        result=m_trade.Sell(lot,price,sl,tp);
//---
    return(result);
}
```


CheckClose

Checks conditions to close position.

```
virtual bool CheckClose()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

1. It checks Expert Advisor Stop Out conditions (CheckClose() method of money management object). If condition is satisfied, it closes the position, deletes all orders ([CloseAll\(\)](#)), and exits.
2. It checks conditions to close long or short position ([CheckCloseLong\(\)](#) or [CheckCloseShort\(\)](#) methods) and if position is closed, it deletes all orders ([DeleteOrders\(\)](#) method).

Implementation

```
//+-----+
//| Check for position close or limit/stop order delete |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckClose()
{
    double lot;
    //--- position must be selected before call
    if((lot=m_money.CheckClose(GetPointer(m_position)))!=0.0)
        return(CloseAll(lot));
    //--- check for position type
    if(m_position.PositionType()==POSITION_TYPE_BUY)
    {
        //--- check the possibility of closing the long position / delete pending orders
        if(CheckCloseLong())
        {
            DeleteOrders();
            return(true);
        }
    }
    else
    {
        //--- check the possibility of closing the short position / delete pending orders
        if(CheckCloseShort())
        {
            DeleteOrders();
            return(true);
        }
    }
    //--- return without operations
    return(false);
}
```


CheckCloseLong

Checks conditions to close a long position.

```
virtual bool CheckCloseLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks conditions to close a long position (CheckCloseLong() method of Signal object) and if they are satisfied, it closes the open position ([CloseLong\(...\)](#) method).

Implementation

```
//+-----+
//| Check for long position close or limit/stop order delete |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckCloseLong()
{
    double price=EMPTY_VALUE;
    ///--- check for long close operations
    if(m_signal.CheckCloseLong(price))
        return(CloseLong(price));
    ///--- return without operations
    return(false);
}
```


CheckCloseShort

Checks conditions to close a short position.

```
virtual bool CheckCloseShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks conditions to close a short position (CheckCloseShort() method of Signal object) and if they are satisfied, it closes the position ([CloseShort\(\)](#) method).

Implementation

```
//+-----+
//| Check for short position close or limit/stop order delete |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckCloseShort()
{
    double price=EMPTY_VALUE;
    ///--- check for short close operations
    if(m_signal.CheckCloseShort(price))
        return(CloseShort(price));
    ///--- return without operations
    return(false);
}
```


CloseAll

It performs partial of full position closing.

```
virtual bool CloseAll(
    double lot // lot
)
```

Parameters

lot

[in] Number of lots to reduce the position.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionClose method is used, which can also be used on accounts with the netting mode. The [DeleteOrders\(\)](#) method is used to delete orders.

Implementation

```
//+-----+
//| Position close and orders delete |
//| INPUT: lot - volume for close. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CloseAll(double lot)
{
    bool result=false;
    //-- check for close operations
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    else
    {
        if(m_position.PositionType()==POSITION_TYPE_BUY)
            result=m_trade.Sell(lot,0,0,0);
        else
            result=m_trade.Buy(lot,0,0,0);
    }
    result|=DeleteOrders();
    //--
    return(result);
}
```


Close

Closes the opened position.

```
virtual bool Close()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

Closes the position (PositionClose() method of CTrade class object).

Implementation

```
//+-----+
//| Position close                               |
//| INPUT:  no.                                  |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                                  |
//+-----+
bool CExpert::Close()
{
    return(m_trade.PositionClose(m_symbol.Name()));
}
```


CloseLong

Closes the long position.

```
virtual bool CloseLong(  
    double price // price  
)
```

Parameters

price

[in] Market entry price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionCloseByTicket method is used.

Implementation

```
//+-----+  
//| Long position close |  
//| INPUT: price - price for close. |  
//| OUTPUT: true-if trade operation processed, false otherwise. |  
//| REMARK: no. |  
//+-----+  
bool CExpert::CloseLong(double price)  
{  
    bool result=false;  
    //---  
    if(price==EMPTY_VALUE)  
        return(false);  
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)  
        result=m_trade.PositionCloseByTicket(m_position.Identifier());  
    else  
        result=m_trade.Sell(m_position.Volume(),price,0,0);  
    //---  
    return(result);  
}
```


CloseShort

Closes the short position.

```
virtual bool CloseShort(  
    double price // price  
)
```

Parameters

price

[in] Market entry price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionCloseByTicket method is used.

Implementation

```
//+-----+  
//| Short position close |  
//| INPUT: price - price for close. |  
//| OUTPUT: true-if trade operation successful, false otherwise. |  
//| REMARK: no. |  
//+-----+  
bool CExpert::CloseShort(double price)  
{  
    bool result=false;  
    //---  
    if(price==EMPTY_VALUE)  
        return(false);  
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)  
        result=m_trade.PositionCloseByTicket(m_position.Identifier());  
    else  
        result=m_trade.Buy(m_position.Volume(),price,0,0);  
    //---  
    return(result);  
}
```


CheckTrailingStop

It checks Trailing Stop conditions of the opened position.

```
virtual bool CheckTrailingStop()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened position ([CheckTrailingStopLong\(\)](#) or [CheckTrailingStopShort\(\)](#) for long and short positions).

Implementation

```
//+-----+
//| Check for trailing stop/profit position |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckTrailingStop()
{
    //--- position must be selected before call
    if(m_position.PositionType()==POSITION_TYPE_BUY)
    {
        //--- check the possibility of modifying the long position
        if(CheckTrailingStopLong()) return(true);
    }
    else
    {
        //--- check the possibility of modifying the short position
        if(CheckTrailingStopShort()) return(true);
    }
    //--- return without operations
    return(false);
}
```


CheckTrailingStopLong

It checks Trailing Stop conditions of the opened long position.

```
virtual bool CheckTrailingStopLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened long position (CheckTrailingStopLong(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters ([TrailingStopLong\(...\)](#) method).

Implementation

```
//+-----+
//| Check for trailing stop/profit long position |
//| INPUT:  no.                                |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                                |
//+-----+
bool CExpert::CheckTrailingStopLong()
{
    double sl=EMPTY_VALUE;
    double tp=EMPTY_VALUE;
    //-- check for long trailing stop operations
    if(m_trailing.CheckTrailingStopLong(GetPointer(m_position),sl,tp))
    {
        if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
        if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
        //-- long trailing stop operations
        return(TrailingStopLong(sl,tp));
    }
    //-- return without operations
    return(false);
}
```


CheckTrailingStopShort

It checks Trailing Stop conditions of the opened short position.

```
virtual bool CheckTrailingStopShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened short position (CheckTrailingStopShort(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters ([TrailingStopShort\(...\)](#) method).

Implementation

```
//+-----+
//| Check for trailing stop/profit short position |
//| INPUT:  no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckTrailingStopShort()
{
    double sl=EMPTY_VALUE;
    double tp=EMPTY_VALUE;
    //-- check for short trailing stop operations
    if(m_trailing.CheckTrailingStopShort(GetPointer(m_position),sl,tp))
    {
        if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
        if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
        //-- short trailing stop operations
        return(TrailingStopShort(sl,tp));
    }
    //-- return without operations
    return(false);
}
```


TrailingStopLong

It modifies parameters of the opened long position.

```
virtual bool TrailingStopLong(  
    double sl,      // Stop Loss  
    double tp,      // Take Profit  
)
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

The function modifies parameters of the opened long position (PositionModify(...) method of CTrade class object).

Implementation

```
//+-----+  
//| Trailing stop/profit long position |  
//| INPUT:  sl - new stop loss,       |  
//|         tp - new take profit.     |  
//| OUTPUT: true-if trade operation successful, false otherwise. |  
//| REMARK: no.                      |  
//+-----+  
bool CExpert::TrailingStopLong(double sl,double tp)  
{  
    return(m_trade.PositionModify(m_symbol.Name(),sl,tp));  
}
```


TrailingStopShort

It modifies parameters of the opened short position.

```
virtual bool TrailingStopLong(  
    double sl,    // Stop Loss  
    double tp,    // Take Profit  
)
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

The function modifies parameters of the opened short position (PositionModify(...) method of CTrade class object).

Implementation

```
//+-----+  
//| Trailing stop/profit short position |  
//| INPUT:  sl - new stop loss,         |  
//|         tp - new take profit.       |  
//| OUTPUT: true-if trade operation successful, false otherwise. |  
//| REMARK: no.                        |  
//+-----+  
bool CExpert::TrailingStopShort(double sl,double tp)  
{  
    return(m_trade.PositionModify(m_symbol.Name(),sl,tp));  
}
```


CheckTrailingOrderLong

Checks Trailing Stop conditions of Buy Limit/Stop trailing order.

```
virtual bool CheckTrailingOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions for buy limit/stop trailing order (CheckTrailingOrderLong() method of Trade Signals object) and modifies the order parameters if necessary ([TrailingOrderLong\(...\)](#) method).

Implementation

```
//+-----+
//| Check for trailing long limit/stop order |
//| INPUT:  no.                             |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                             |
//+-----+
bool CExpert::CheckTrailingOrderLong()
{
    double price;
    //--- check the possibility of modifying the long order
    if(m_signal.CheckTrailingOrderLong(GetPointer(m_order),price))
        return(TrailingOrderLong(m_order.PriceOpen()-price));
    //--- return without operations
    return(false);
}
```


CheckTrailingOrderShort

It checks Trailing Stop conditions of Sell Limit/Stop trailing order.

```
virtual bool CheckTrailingOrderShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions for sell limit/stop trailing order (CheckTrailingOrderShort() method of Trade Signals object) and modifies the order parameters if necessary ([TrailingOrderShort\(\)](#) method).

Implementation

```
//+-----+
//| Check for trailing short limit/stop order |
//| INPUT:  no.                             |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                             |
//+-----+
bool CExpert::CheckTrailingOrderShort()
{
    double price;
    ///--- check the possibility of modifying the short order
    if(m_signal.CheckTrailingOrderShort(GetPointer(m_order),price))
        return(TrailingOrderShort(m_order.PriceOpen()-price));
    ///--- return without operations
    return(false);
}
```


TrailingOrderLong

It modifies parameters of Buy Limit/Stop trailing order.

```
virtual bool TrailingOrderLong(
    double delta // delta
)
```

Parameters

delta

[in] Price delta.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It modifies parameters of Buy Limit/Stop trailing order (OrderModify(...) method of CTrade class object).

Implementation

```
//+-----+
//| Trailing long limit/stop order |
//| INPUT: delta - price change. |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::TrailingOrderLong(double delta)
{
    ulong ticket=m_order.Ticket();
    double price =m_order.PriceOpen()-delta;
    double sl =m_order.StopLoss()-delta;
    double tp =m_order.TakeProfit()-delta;
    //--- modifying the long order
    return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpir
}
```


TrailingOrderShort

It modifies parameters of Sell Limit/Stop trailing order.

```
virtual bool TrailingOrderShort(
    double delta // delta
)
```

Parameters

delta

[in] Price delta.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It modifies parameters of Sell Limit/Stop trailing order (OrderModify(...) method of CTrade class object).

Implementation

```
//+-----+
//| Trailing short limit/stop order |
//| INPUT: delta - price change. |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::TrailingOrderShort(double delta)
{
    ulong ticket=m_order.Ticket();
    double price =m_order.PriceOpen()-delta;
    double sl =m_order.StopLoss()-delta;
    double tp =m_order.TakeProfit()-delta;
    //--- modifying the short order
    return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpiration));
}
```


CheckDeleteOrderLong

It checks conditions to delete Buy Limit/Stop order.

```
virtual bool CheckDeleteOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks the order expiration time. It checks conditions to delete the Buy Limit/Stop order (CheckCloseLong(...) method of Signal class object) and deletes the order if condition is satisfied ([DeleteOrderLong\(\)](#) method).

Implementation

```
//+-----+
//| Check for delete long limit/stop order |
//| INPUT: no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckDeleteOrderLong()
{
    double price;
    //-- check the possibility of deleting the long order
    if(m_expiration!=0 && TimeCurrent()>m_expiration)
    {
        m_expiration=0;
        return(DeleteOrderLong());
    }
    if(m_signal.CheckCloseLong(price))
        return(DeleteOrderLong());
    //-- return without operations
    return(false);
}
```


CheckDeleteOrderShort

It checks conditions to delete Sell Limit/Stop order.

```
virtual bool CheckDeleteOrderShort ()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

1. It checks the order expiration time.
2. It checks conditions to delete the Sell Limit/Stop order (CheckCloseShort(...) method of Signal class object) and deletes the order if one of the conditions is satisfied ([DeleteOrderShort\(\)](#) method).

Implementation

```
//+-----+
//| Check for delete short limit/stop order |
//| INPUT: no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckDeleteOrderShort ()
{
    double price;
    //-- check the possibility of deleting the short order
    if(m_expiration!=0 && TimeCurrent()>m_expiration)
    {
        m_expiration=0;
        return(DeleteOrderShort());
    }
    if(m_signal.CheckCloseShort(price))
        return(DeleteOrderShort());
    //-- return without operations
    return(false);
}
```


DeleteOrders

Deletes all orders.

```
virtual bool DeleteOrders()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes all orders ([DeleteOrder\(\)](#) for all orders).

Implementation

```
//+-----+
//| Delete all limit/stop orders |
//| INPUT: no. |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::DeleteOrders()
{
    bool result=false;
    int total=OrdersTotal();
    //---
    for(int i=total-1;i>=0;i--)
    {
        if(m_order.Select(OrderGetTicket(i)))
        {
            if(m_order.Symbol()!=m_symbol.Name()) continue;
            result|=DeleteOrder();
        }
    }
    //---
    return(result);
}
```


DeleteOrder

Deletes the Limit/Stop order.

```
virtual bool DeleteOrder()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes the Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+-----+
//| Delete limit/stop order                               |
//| INPUT:  no.                                           |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no.                                           |
//+-----+
bool CExpert::DeleteOrder()
{
    return(m_trade.OrderDelete(m_order.Ticket()));
}
```


DeleteOrderLong

Deletes the Buy Limit/Stop order.

```
virtual bool DeleteOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes Buy Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+-----+
//| Delete long limit/stop order |
//| INPUT:  no.                  |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no.                  |
//+-----+
bool CExpert::DeleteOrderLong()
{
    return(m_trade.OrderDelete(m_order.Ticket()));
}
```


DeleteOrderShort

Deletes the Sell Limit/Stop order.

```
virtual bool DeleteOrderShort ()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes the Sell Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+-----+  
//| Delete short limit/stop order |  
//| INPUT:  no.                   |  
//| OUTPUT: true-if trade operation successful, false otherwise. |  
//| REMARK: no.                   |  
//+-----+  
bool CExpert::DeleteOrderShort ()  
{  
    return(m_trade.OrderDelete(m_order.Ticket()));  
}
```


LotOpenLong

Gets trade volume for buy operation.

```
double LotOpenLong(  
    double price, // price  
    double sl      // Stop Loss  
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for buy operation.

Note

It gets trade volume for buy operation (CheckOpenLong(...) method of money management object).

Implementation

```
//+-----+  
//| Method of getting the lot for open long position. |  
//| INPUT:  price - price,                             |  
//|        sl    - stop loss.                         |  
//| OUTPUT: lot for open.                             |  
//| REMARK: no.                                       |  
//+-----+  
double CExpert::LotOpenLong(double price,double sl)  
{  
    return(m_money.CheckOpenLong(price,sl));  
}
```


LotOpenShort

Gets trade volume for sell operation.

```
double LotOpenShort(  
    double price, // price  
    double sl      // Stop Loss  
)
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for sell operation.

Note

It gets trade volume for sell operation (CheckOpenShort(...) method of money management object).

Implementation

```
//+-----+  
//| Method of getting the lot for open short position. |  
//| INPUT:  price - price, |  
//|        sl    - stop loss. |  
//| OUTPUT: lot for open. |  
//| REMARK: no. |  
//+-----+  
double CExpert::LotOpenShort(double price,double sl)  
{  
    return(m_money.CheckOpenShort(price,sl));  
}
```


LotReverse

Gets trade volume for position reverse.

```
double LotReverse(  
    double    sl      // Stop Loss  
)
```

Parameters

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for position reverse operation.

Note

It gets trade volume for position reverse operation (CheckReverse(...) method of money management object).

Implementation

```
//+-----+  
//| Method of getting the lot for reverse position. |  
//| INPUT:  sl - stop loss.                        |  
//| OUTPUT: lot for open.                          |  
//| REMARK: no.                                    |  
//+-----+  
double CExpert::LotReverse(double sl)  
{  
    return(m_money.CheckReverse(GetPointer(m_position),sl));  
}
```


PrepareHistoryDate

Sets starting date for tracking of trade history.

```
void PrepareHistoryDate()
```

Note

The trade history tracking period is set from the beginning of the month (but not less than one day).

HistoryPoint

Creates a checkpoint of trade history.

```
void HistoryPoint(  
    bool    from_check_trade=false    // flag  
)
```

Parameters

from_check_trade=false
[in] Flag to avoid recursion.

Note

It saves the amount of positions, orders, deals, and historical orders.

CheckTradeState

Compares the current state with the saved one and calls the corresponding event handler.

```
bool CheckTradeState ()
```

Return Value

true - event has been handled, otherwise - false.

Note

It checks the number of positions, orders, deals, and historical orders by comparing with the values saved by [HistoryPoint\(\)](#) method. If trade history has changed, it calls the corresponding virtual event handler.

WaitEvent

Sets the event waiting flag.

```
void WaitEvent(  
    ENUM_TRADE_EVENTS  event      // flag  
)
```

Parameters

event

[in] Flag of waiting for an event (from ENUM_TRADE_EVENTS enumeration) to set.

Return Value

None.

Event flags

```
///--- flags of expected events  
enum ENUM_TRADE_EVENTS  
{  
    TRADE_EVENT_NO_EVENT           =0,          // no expected events  
    TRADE_EVENT_POSITION_OPEN      =0x1,        // flag of expecting the "opening of  
    TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,    // flag of expecting of the "modifica  
    TRADE_EVENT_POSITION_MODIFY     =0x4,        // flag of expecting of the "modifica  
    TRADE_EVENT_POSITION_CLOSE      =0x8,        // flag of expecting of the "closing  
    TRADE_EVENT_POSITION_STOP_TAKE  =0x10,       // flag of expecting of the "trigger  
    TRADE_EVENT_ORDER_PLACE         =0x20,       // flag of expecting of the "placing  
    TRADE_EVENT_ORDER_MODIFY        =0x40,       // flag of expecting of the "modifica  
    TRADE_EVENT_ORDER_DELETE        =0x80,       // flag of expecting of the "deletio  
    TRADE_EVENT_ORDER_TRIGGER       =0x100      // flag of expecting of the "trigger  
};
```


NoWaitEvent

Resets the event waiting flag.

```
void NoWaitEvent(  
    ENUM_TRADE_EVENTS    event    // flag  
)
```

Parameters

event

[in] Flag with events (from ENUM_TRADE_EVENTS enumeration) to reset.

Return Value

None.

Event flags

```
///--- flags of expected events  
enum ENUM_TRADE_EVENTS  
{  
    TRADE_EVENT_NO_EVENT                =0,           // no expected events  
    TRADE_EVENT_POSITION_OPEN           =0x1,         // flag of expecting the "opening of  
    TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,         // flag of expecting of the "modifica  
    TRADE_EVENT_POSITION_MODIFY         =0x4,         // flag of expecting of the "modifica  
    TRADE_EVENT_POSITION_CLOSE          =0x8,         // flag of expecting of the "closing  
    TRADE_EVENT_POSITION_STOP_TAKE      =0x10,        // flag of expecting of the "trigger  
    TRADE_EVENT_ORDER_PLACE             =0x20,        // flag of expecting of the "placing  
    TRADE_EVENT_ORDER_MODIFY            =0x40,        // flag of expecting of the "modifica  
    TRADE_EVENT_ORDER_DELETE            =0x80,        // flag of expecting of the "deletio  
    TRADE_EVENT_ORDER_TRIGGER           =0x100        // flag of expecting of the "trigger  
};
```


TradeEventPositionStopTake

Event handler of the "Position Stop Loss/Take Profit triggered" event.

```
virtual bool TradeEventPositionStopTake()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventOrderTriggered

Event handler of the "Pending Order triggered" event.

```
virtual bool TradeEventOrderTriggered()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventPositionOpened

Event handler of the "Position opened" event.

```
virtual bool TradeEventPositionOpened()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventPositionVolumeChanged

Event handler of the "Position volume changed" event.

```
virtual bool TradeEventPositionVolumeChanged()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventPositionModified

Event handler of the "Position modified" event.

```
virtual bool TradeEventPositionModified()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventPositionClosed

Event handler of the "Position closed" event.

```
virtual bool TradeEventPositionClosed()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventOrderPlaced

Event handler of the "Pending order placed" event.

```
virtual bool TradeEventOrderPlaced()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventOrderModified

Event handler of the "Pending order modified" event.

```
virtual bool TradeEventOrderModified()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventOrderDeleted

Event handler of the "Pending order deleted" event.

```
virtual bool TradeEventOrderDeleted()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

TradeEventNotIdentified

Event handler of the non-identified event.

```
virtual bool TradeEventNotIdentified()
```

Return Value

The [CExpert](#) class method does nothing and always returns true.

Note

Note that several trade events can arrive, in such cases it is difficult to identify them.

TimeframeAdd

Add a timeframe for tracking.

```
void TimeframeAdd(  
    ENUM_TIMEFRAMES    period        // timeframe  
)
```

Parameters

period

[in] Timeframe (from [ENUM_TIMEFRAMES](#) enumeration) to be tracked.

Return Value

None.

TimeframesFlags

Forms the timeframe flags.

```
int TimeframesFlags(  
    MqlDateTime&    time           // reference  
)
```

Parameters

time

[in] Reference to [MqlDateTime](#) type structure containing a new time.

Return Value

It returns the flag that indicates timeframes with a new bar.

CExpertSignal

CExpertSignal is a base class for trading signals, it does nothing (except [CheckReverseLong\(\)](#) and [CheckReverseShort\(\)](#) methods) but provides the interfaces.

How to use it:

1. Prepare an algorithm for trading signals;
2. Create your own trading signal class, inherited from CExpertSignal class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trading signal classes in the Expert\Signal\ folder.

Description

CExpertSignal is a base class for implementation of trading signal algorithms.

Declaration

```
class CExpertSignal : public CExpertBase
```

Title

```
#include <Expert\ExpertSignal.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

CExpertSignal

Direct descendants

CSignalAC, CSignalAMA, CSignalAO, CSignalBearsPower, CSignalBullsPower, CSignalCCI, CSignalDeM, CSignalDEMA, CSignalEnvelopes, CSignalFrAMA, CSignalRSI, CSignalRVI, CSignalSAR, CSignalStoch, CSignalTEMA, CSignalTriX, CSignalWPR

Class Methods by Groups

Initialization	
virtual InitIndicators	Initializes indicators and timeseries
virtual ValidationSettings	Checks the object settings
virtual AddFilter	Adds a filter to combined signal
Access to Protected Data	
BasePrice	Sets base price level
UsedSeries	Gets the flags of timeseries used
Parameters Setting	

Weight	Sets the value of "Weight" parameter
PatternsUsage	Sets the value of "PatternsUsage" parameter
General	Sets the value of "General" parameter
Ignore	Sets the value of "Ignore" parameter
Invert	Sets the value of "Invert" parameter
ThresholdOpen	Sets the value of "ThresholdOpen" parameter
ThresholdClose	Sets the value of "ThresholdClose" parameter
PriceLevel	Sets the value of "PriceLevel" parameter
StopLevel	Sets the value of "StopLevel" parameter
TakeLevel	Sets the value of "TakeLevel" parameter
Expiration	Sets the value of "Expiration" parameter
Magic	Sets the value of "Magic" parameter
Checking Trading Conditions	
virtual CheckOpenLong	Checks conditions to open long position
virtual CheckCloseLong	Checks conditions to close long position
virtual CheckOpenShort	Checks conditions to open short position
virtual CheckCloseShort	Checks conditions to close short position
virtual CheckReverseLong	Checks conditions of long position reversal
virtual CheckReverseShort	Checks conditions of short position reversal
Trade Parameters Setting	
virtual OpenLongParams	Sets parameters for long position opening
virtual OpenShortParams	Sets parameters for short position opening
virtual CloseLongParams	Sets parameters for long position closing
virtual CloseShortParams	Sets parameters for short position closing
Checking of Order Trailing Conditions	
virtual CheckTrailingOrderLong	Checks conditions to modify parameters of Buy Pending order
virtual CheckTrailingOrderShort	Checks conditions to modify parameters of Sell Pending order
Methods to Check Formation of Market Orders	
virtual LongCondition	Gets the result of checking buy conditions

virtual ShortCondition	Gets the result of checking sell conditions
virtual Direction	Gets the "weighted" direction of price

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

BasePrice

Sets base price level.

```
void BasePrice(  
    double    value        // value  
)
```

Parameters

value

[in] New value of the parameter.

Return Value

None.

UsedSeries

Gets the flags of the timeseries used.

```
int BasePrice()
```

Return Value

Flags of the used timeseries (if the symbol/timeframe corresponds to the working symbol/timeframe), otherwise 0.

Weight

Sets new value of "Weight" parameter.

```
void Weight(  
    double    value    // value  
)
```

Parameters

value

[in] "Weight" parameter.

Return Value

None.

PatternUsage

Sets new value of "PatternsUsage" parameter.

```
void PatternUsage(  
    double    value        // value  
)
```

Parameters

value

[in] New value of "PatternsUsage".

Return Value

None.

General

Sets new value of "General" parameter.

```
void General (
    int    value      // value
)
```

Parameters

value

[in] New value of "General".

Return Value

None.

Ignore

Sets new value of "Ignore" parameter.

```
void Ignore(  
    long    value    // value  
)
```

Parameters

value

[in] New value of "Ignore".

Return Value

None.

Invert

Sets new value of "Invert" parameter.

```
void Invert(  
    long    value    // value  
)
```

Parameters

value

[in] New value of "Invert".

Return Value

None.

ThresholdOpen

Sets new value of "ThresholdOpen" parameter.

```
void ThresholdOpen(  
    long    value    // value  
)
```

Parameters

value

[in] New value of "ThresholdOpen".

Return Value

None.

Note

The range of "ThresholdOpen" parameter is from 0 to 100. Used when "voting" to open position.

ThresholdClose

Sets new value of "ThresholdClose" parameter.

```
void ThresholdOpen(  
    long    value    // value  
)
```

Parameters

value

[in] New value of "ThresholdClose".

Return Value

None.

Note

The range of "ThresholdClose" parameter is from 0 to 100. Used when "voting" to close position.

PriceLevel

Sets new value of "PriceLevel" parameter.

```
void PriceLevel(  
    double    value        // value  
)
```

Parameters

value

[in] New value of "PriceLevel".

Return Value

None.

Note

The value of "PriceLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "PriceLevel" is used to define the open price relative to the base price.

StopLevel

Sets new value of "StopLevel" parameter.

```
void StopLevel(  
    double    value        // value  
)
```

Parameters

value

[in] New value of "StopLevel".

Return Value

None.

Note

The value of "StopLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "StopLevel" is used to define the Stop Loss price relative to the open price.

TakeLevel

Sets new value of "TakeLevel" parameter.

```
void TakeLevel(  
    double    value        // value  
)
```

Parameters

value

[in] New value of "TakeLevel".

Return Value

None.

Note

The value of "TakeLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "TakeLevel" is used to define the Take Profit price relative to the open price.

Expiration

Sets the value of "Expiration" parameter.

```
void Expiration(  
    int    value        // value  
)
```

Parameters

value

[in] New value of "Expiration".

Return Value

None.

Note

The value of "Expiration" parameter is defined in bars. It is used as Expiration time for Pending Orders (when trading using pending orders).

Magic

Sets the value of "Magic" parameter.

```
void Magic(  
    int    value    // value  
)
```

Parameters

value

[in] New value of "Magic" (Expert Advisor ID).

Return Value

None.

ValidationSettings

Checks the object settings.

```
virtual bool ValidationSettings()
```

Return Value

true - object settings are correct, otherwise - false.

InitIndicators

Initializes all necessary indicators and timeseries.

```
virtual bool InitIndicators(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful completion, otherwise - false.

Note

The necessary timeseries are initialized only if the object uses the symbol or timeframe different from the one defined at initialization.

AddFilter

Adds a filter to the composite signal.

```
virtual bool InitIndicators(  
    CExpertSignal* filter    // pointer  
)
```

Parameters

indicators

[in] Pointer to filter object.

Return Value

true - successful, otherwise - false.

CheckOpenLong

Checks conditions to open a long position.

```
virtual bool CheckOpenLong(  
    double& price,           // price  
    double& sl,              // Stop Loss  
    double& tp,              // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckOpenShort

Checks conditions to open a short position.

```
virtual bool CheckOpenShort(  
    double& price,           // price  
    double& sl,              // Stop Loss  
    double& tp,              // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

OpenLongParams

Sets parameters to open a long position.

```
virtual bool OpenLongParams(  
    double& price,           // price  
    double& sl,             // Stop Loss  
    double& tp,             // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - successful, otherwise - false.

OpenShortParams

Sets parameters to open a short position.

```
virtual bool OpenShortParams(  
    double& price,           // price  
    double& sl,              // Stop Loss  
    double& tp,              // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - successful, otherwise - false.

CheckCloseLong

Checks conditions to close a long position.

```
virtual bool CheckCloseLong(  
    double& price    // price  
)
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

CheckCloseShort

Checks conditions to close a short position.

```
virtual bool CheckCloseShort(  
    double& price    // price  
)
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

CloseLongParams

Sets parameters to close a long position.

```
virtual bool CloseLongParams(  
    double& price // price  
)
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - successful, otherwise - false.

CloseShortParams

Sets parameters to close a short position.

```
virtual bool CloseShortParams (  
    double& price // price  
)
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - successful, otherwise - false.

CheckReverseLong

Checks conditions of a long position reversal.

```
virtual bool CheckReverseLong(  
    double& price,           // price  
    double& sl,              // Stop Loss  
    double& tp,              // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckReverseShort

Checks conditions of a short position reversal.

```
virtual bool CheckReverseShort(  
    double& price,           // price  
    double& sl,              // Stop Loss  
    double& tp,              // Take Profit  
    datetime& expiration    // expiration  
)
```

Parameters

price

[in][out] Variable for reversal price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckTrailingOrderLong

Checks conditions to modify parameters of Buy Pending order.

```
virtual bool CheckTrailingOrderLong(  
    COrderInfo*   order,          // order  
    double&       price           // price  
)
```

Parameters

order

[in] Pointer to [COrderInfo](#) class object.

price

[in][out] Variable for Stop Loss price.

Return Value

true - condition is satisfied, otherwise - false.

CheckTrailingOrderShort

Checks conditions to modify parameters of Sell Pending order.

```
virtual bool CheckTrailingOrderShort(  
    COrderInfo*    order,          // order  
    double&        price           // price  
)
```

Parameters

order

[in] Pointer to [COrderInfo](#) class object.

price

[in][out] Variable for Stop Loss price.

Return Value

true - condition is satisfied, otherwise - false.

LongCondition

Checks conditions to open a long position.

```
virtual int LongCondition()
```

Return Value

If the conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal). If there is no signal to open a long position, it returns 0.

Note

The base class has no implementation of checking conditions to open a long position and always returns 0.

ShortCondition

Checks conditions to open a short position.

```
virtual int ShortCondition()
```

Return Value

If the conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal). If there isn't a signal to open short position, it returns 0.

Note

The base class has no implementation of checking conditions to open a short position and always returns 0.

Direction

Returns the value of "weighted" price direction.

```
virtual double Direction()
```

Return Value

It returns the value >0 when upward direction is most probable and <0 in case of a downward direction. The absolute value depends on the "strength" of a signal.

Note

If the built-in filters are used, their results are considered when defining the general direction.

CExpertTrailing

CExpertTrailing is a base class for trailing algorithms, it does nothing but provides the interfaces.

How to use it:

1. Prepare an algorithm for trailing;
2. Create your own trailing class inherited from CExpertTrailing class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trailing classes in the Expert\Trailing\ folder.

Description

CExpertTrailing is a base class for implementation of trailing stop algorithms.

Declaration

```
class CExpertTrailing : public CExpertBase
```

Title

```
#include <Expert\ExpertTrailing.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

CExpertTrailing

Direct descendants

[CTrailingFixedPips](#), [CTrailingMA](#), [CTrailingNone](#), [CTrailingPSAR](#)

Class Methods by Groups

Checking of Trailing Stop Conditions	
virtual CheckTrailingStopLong	Checks conditions to modify parameters of the long position
virtual CheckTrailingStopShort	Checks conditions to modify parameters of the short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

CheckTrailingStopLong

Checks conditions to modify parameters of a long position.

```
virtual bool CheckTrailingStopLong(  
    CPositionInfo* position,    // pointer  
    double& sl,                // Stop Loss  
    double& tp                 // Take Profit  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

Note

Base class method always returns false.

CheckTrailingStopShort

Checks conditions to modify parameters of a short position.

```
virtual bool CheckTrailingStopShort (
    CPositionInfo* position,    // pointer
    double& sl,                // Stop Loss
    double& tp                 // Take Profit
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

Note

Base class method always returns false.

CExpertMoney

CExpertMoney is a base class for money and risk management algorithms.

Description

CExpertMoney is a base class for implementation of money and risk management classes.

Declaration

```
class CExpertMoney : public CObject
```

Title

```
#include <Expert\ExpertMoney.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

CExpertMoney

Direct descendants

[CMoneyFixedLot](#), [CMoneyFixedMargin](#), [CMoneyFixedRisk](#), [CMoneyNone](#), [CMoneySizeOptimized](#)

Class Methods by Groups

Access to Protected Data	
Percent	Sets the value of "Risk percent" parameter
Initialization	
virtual ValidationSettings	Checks the settings
Checking Trading Conditions	
virtual CheckOpenLong	Gets the volume for a long position
virtual CheckOpenShort	Gets the volume for a short position
virtual CheckReverse	Gets the volume for a reverse of the position
virtual CheckClose	Checks conditions to close an opened position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Percent

Sets the value of "Risk percent" parameter.

```
void Percent(  
    double percent    // risk percent  
)
```

Parameters

percent

[in] Risk percent.

Return Value

None.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

Base class method always returns true.

CheckOpenLong

Gets the volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss  
)
```

Parameters

price

[in] Opening price of a long position.

sl

[in] Stop Loss price of a long position.

Return Value

Trade volume for a long position.

CheckOpenShort

Gets the volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss  
)
```

Parameters

price

[in] Opening price for a short position.

sl

[in] Stop Loss price of a short position.

Return Value

Trade volume for a short position.

CheckReverse

Gets the volume for reverse of the position.

```
virtual double CheckReverse(  
    CPositionInfo* position,    // pointer  
    double sl                  // Stop Loss  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) class object.

sl

[in] Stop Loss price.

Return Value

Volume for reverse of the position.

CheckClose

Checks conditions to close the opened position.

```
virtual double CheckClose()
```

Return Value

true - condition is satisfied, otherwise - false.

Modules of Trade Signals

The standard delivery of the client terminal includes a set of ready-made modules of trade signals for "MQL5 Wizard". When creating an Expert Advisor in MQL5 Wizard, you can use any combination of the modules of trade signals (up to 64). The final decision on a trade operation is made on the basis of complex analysis of signals obtained from all included modules. The detailed description of the mechanism of making trade decisions is given [below](#).

The standard delivery includes the following modules of signals:

- [Signals of the Indicator Accelerator Oscillator](#)
- [Signals of the Indicator Adaptive Moving Average](#)
- [Signals of the Indicator Awesome Oscillator](#)
- [Signals of the Oscillator Bears Power](#)
- [Signals of the Oscillator Bulls Power](#)
- [Signals of the Oscillator Commodity Channel Index](#)
- [Signals of the Oscillator DeMarker](#)
- [Signals of the Indicator Double Exponential Moving Average](#)
- [Signals of the Indicator Envelopes](#)
- [Signals of the Indicator Fractal Adaptive Moving Average](#)
- [Signals of the Intraday Time Filter](#)
- [Signals of the Oscillator MACD](#)
- [Signals of the Indicator Moving Average](#)
- [Signals of the Indicator Parabolic SAR](#)
- [Signals of the Oscillator Relative Strength Index](#)
- [Signals of the Oscillator Relative Vigor Index](#)
- [Signals of the Oscillator Stochastic](#)
- [Signals of the Oscillator Triple Exponential Average](#)
- [Signals of the Indicator Triple Exponential Moving Average](#)
- [Signals of the Oscillator Williams Percent Range](#)

The Mechanism of Making Trade Decisions on the Basis of Signal Modules

The mechanism of making trade decisions can be represented as the following list of basic principles:

- Each of the modules of signals has its set of market modules (certain combination of prices and values of an indicator).
- Each market model has a significance that may vary with the range of 1 to 100. The higher is the significance, the stronger the model is.
- Each of the models generates a forecast of direction of the price movement.
- A forecast of a module is the result of search for embedded models, and it is outputted as a number within the range of -100 to 100. The sign determines the direction of forecast movement (negative

sign means the price will fall, positive sign means the price will rise). The absolute value corresponds to the strength of the best found model.

- The forecast of each module is sent to the final "voting" with a weight coefficient of 0 to 1 specified in its settings ("Weight").
- The result of voting is a number within the range of -100 to 100, where the sign determines direction of the forecast movement, and the absolute value characterizes the strength of the signal. It is calculated as the Arithmetic mean of weighted forecasts of all the modules of signals. The absolute value is used by an Expert Advisor to make trade decisions.

Each generated Expert Advisor has two adjustable settings – threshold levels of opening and closing a position (ThresholdOpen and ThresholdClose) that can be equal to a value in the range of 0 to 100. If the strength of final signal exceeds a threshold level, a trade operation that corresponds to the sign of the signal is performed.

Examples

Consider an Expert Advisor with the following threshold levels: ThresholdOpen=20 and ThresholdClose=90. Two modules of signals participate in making decisions on trade operations: the [MA](#) module with weight 0.4 and the [Stochastic](#) module with weight 0.8. Let's analyze two variants of obtained trade signals:

Variant 1

The price crossed the rising MA upwards. This case corresponds to one of the market models implemented in the [MA module](#). This model implies a rise of price. Its significance is equal to 100. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the [Stochastic module](#). This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as $0.4 * 100 = 40$. The value from the Stochastic module is calculated as $0.8 * (-80) = -64$. The final value is calculated as the Arithmetic mean of these two rates: $(40 - 64)/2 = -12$. The result of voting is the signal for selling with relative strength equal to 12. The threshold level that is equal to 20 is not reached. Thus a trade operation is not performed.

Variant 2

The price crossed the rising MA downwards. This case corresponds to one of the models implemented in the [MA module](#). This model implies a rise of price. Its significance is equal to 10. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the [Stochastic module](#). This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as $0.4 * 10 = 4$. The value from the Stochastic module is calculated as $0.8 * (-80) = -64$. The final value is calculated as the Arithmetic mean of these two rates: $(4 - 64)/2 = -30$. The result of voting is the signal for selling with relative strength equal to 30. The threshold level that is equal to 20 is reached. Thus the result is the signal for opening a short position.



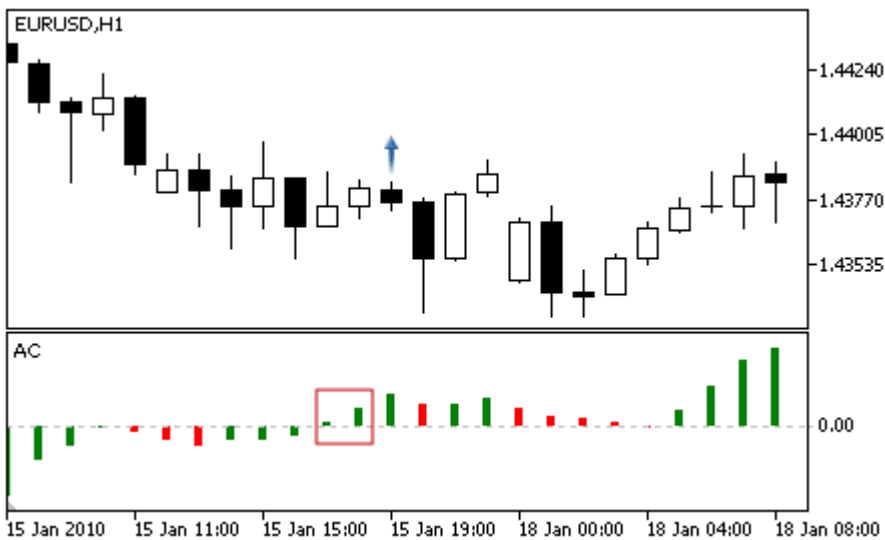
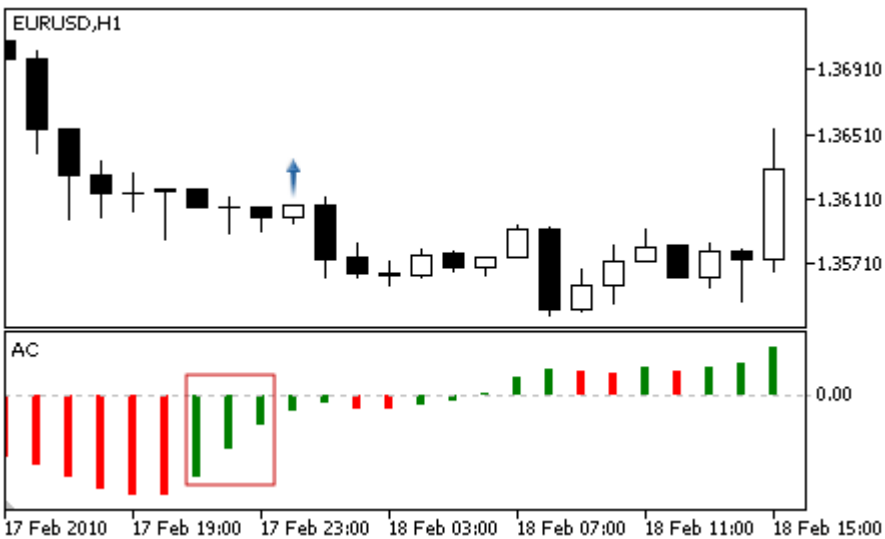
- a) Divergence of the price and the Stochastic oscillator (variants 1 and 2).
- b) The price crossed the MA indicator upwards (variant 1).
- c) The price crossed the MA indicator downwards (variant 2).

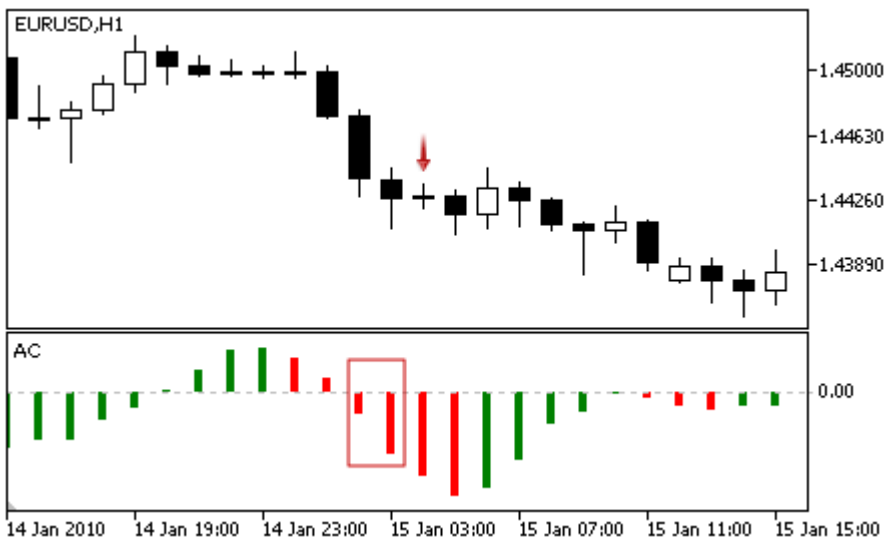
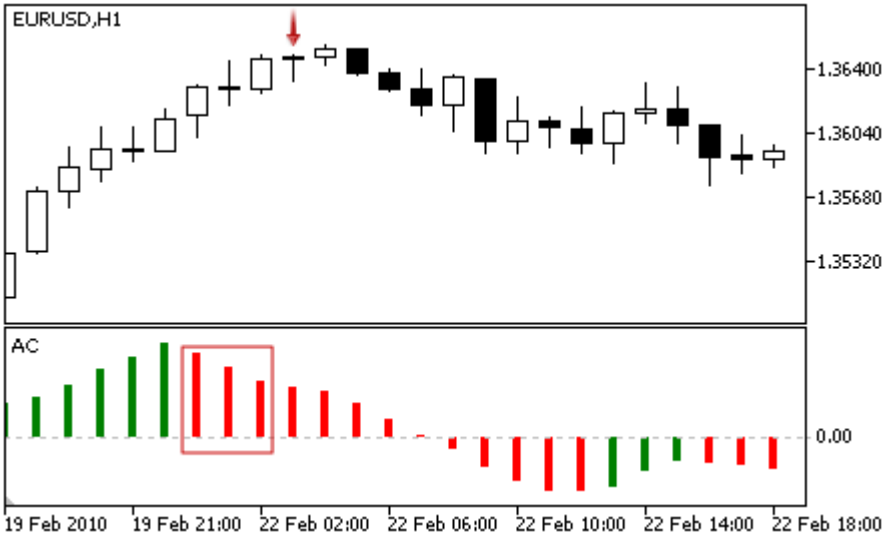
Signals of the Indicator Accelerator Oscillator

This module is based on the market models of the indicator [Accelerator Oscillator](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> The indicator value is above 0 and it rises at the analyzed and at the previous bars.  <ul style="list-style-type: none"> The indicator value is below 0 and it rises at the analyzed and at the previous bars. 
For selling	<ul style="list-style-type: none"> The indicator value is below 0 and it falls at the analyzed and at the previous bars.

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> The indicator value is above 0 and it falls at the analyzed and at the two previous bars. 
No objections to buying	The indicator value grows at the analyzed bar.
No objections to selling	The indicator value falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only"), an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

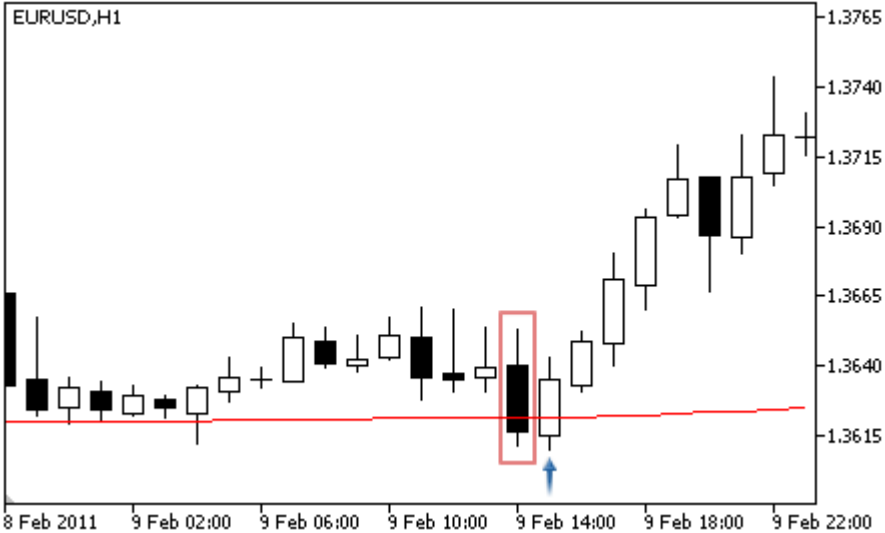
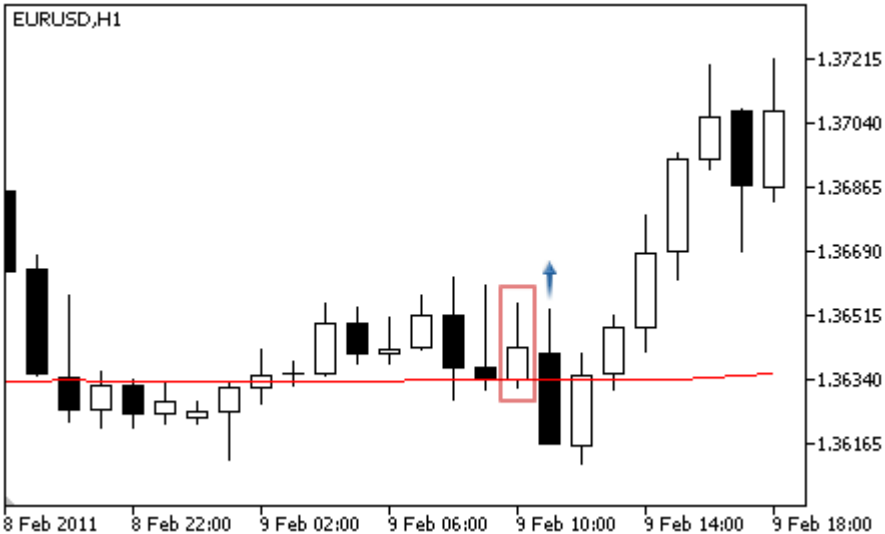
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

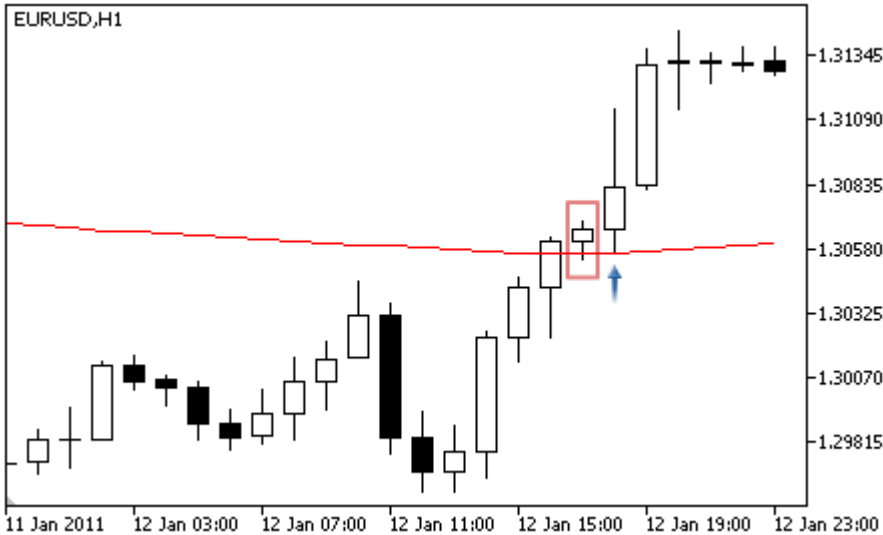
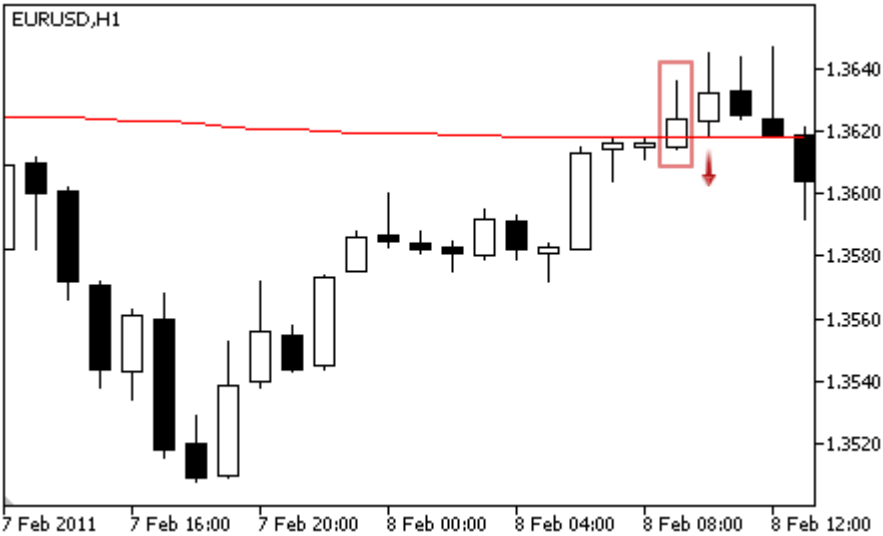
Signals of the Indicator Adaptive Moving Average

This module is based on the market models of the indicator [Adaptive Moving Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Failed breakout. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) but the indicator rises (weak indicator line roll-back signal).  Moving Average crossover. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • True breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (indicator line roll-back signal). 
For selling	<ul style="list-style-type: none"> • Failed breakout. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) but the indicator falls (weak indicator line roll-back signal).  <ul style="list-style-type: none"> • Moving Average crossover. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> • True breakout. The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (indicator line roll-back signal). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only"), an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

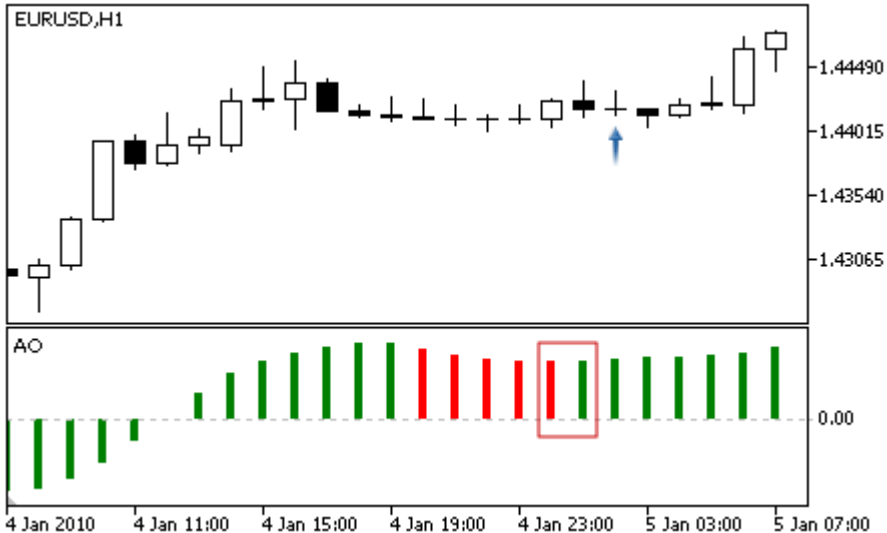
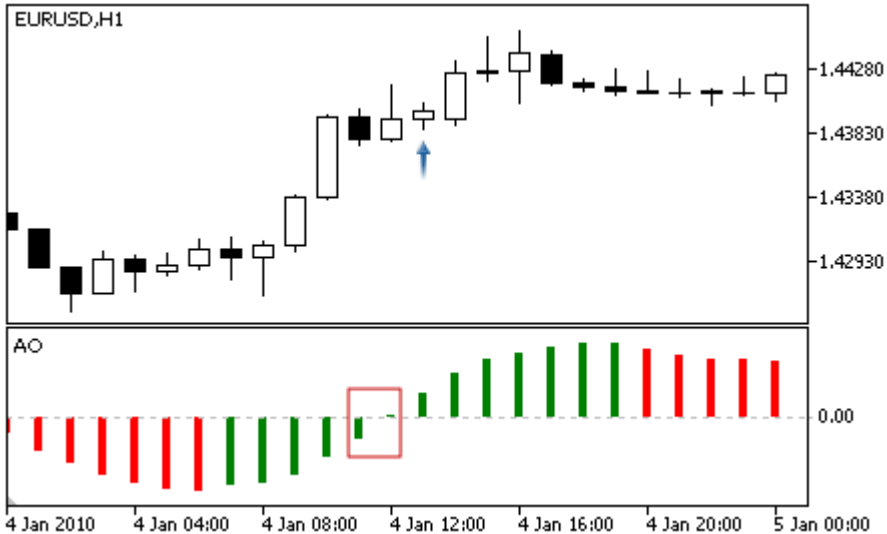
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging .
Applied	A price series used for calculation of the indicator.

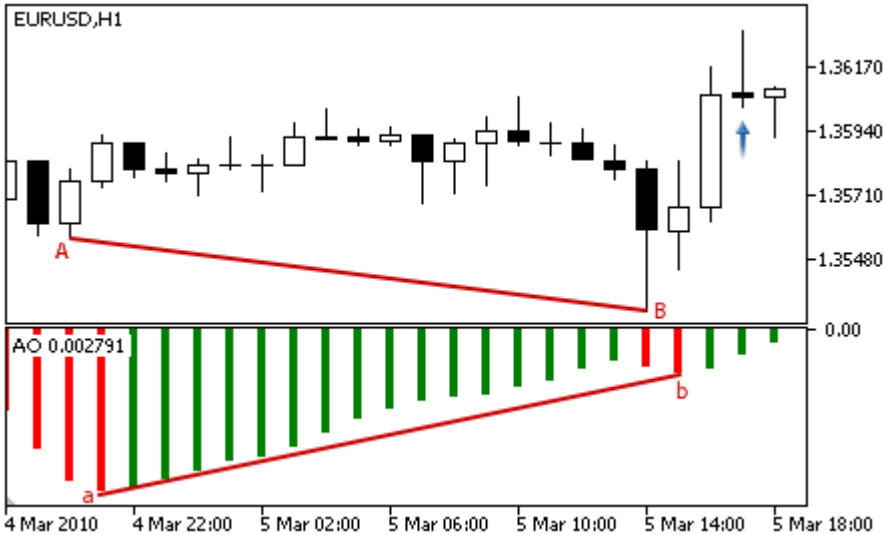
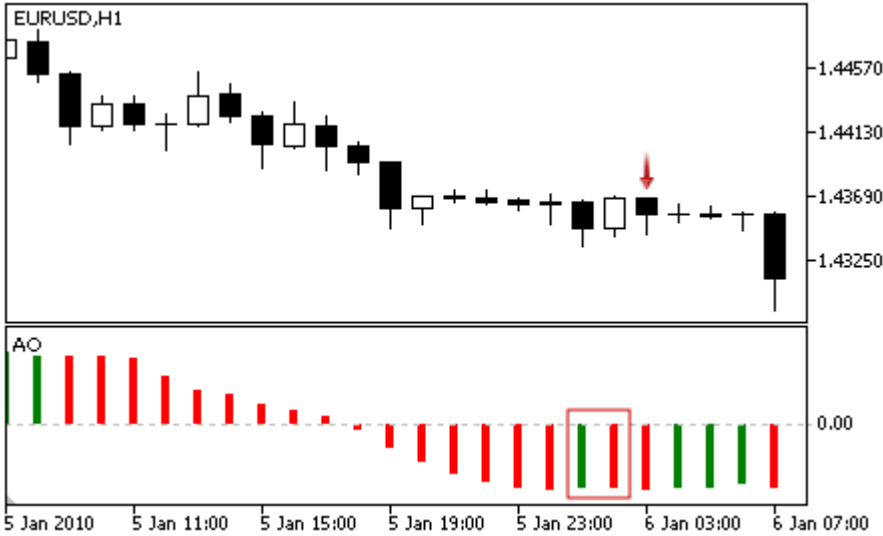
Signals of the Indicator Awesome Oscillator

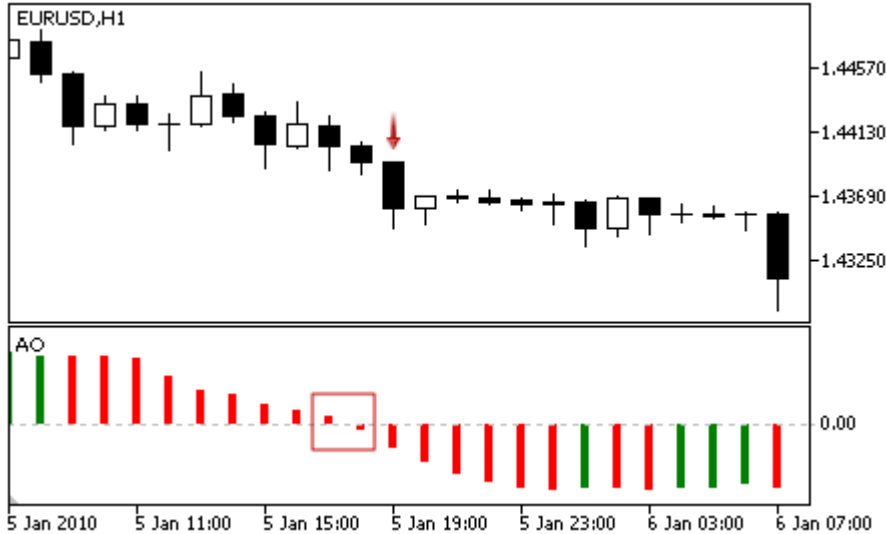
This module of signals is based on the market models of the indicator [Awesome Oscillator](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Saucer – value of the indicator at the analyzed bar rises, and it fell at the previous bars; at that, both values are above 0.  Crossing the zero line – value of the indicator is above 0 at the analyzed bar, and it is below 0 at the previous bar. 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Divergence — the first analyzed bottom of the indicator is shallower than the previous one, and the corresponding price bottom is deeper than the previous one. In addition, the indicator must not rise above the zero level. 
For selling	<ul style="list-style-type: none"> • Saucer — value of the indicator at the analyzed bar falls, and it rose at the previous bars; at that, both values are below 0.  <ul style="list-style-type: none"> • Crossing the zero line — value of the indicator is below 0 at the analyzed bar, and it is above 0 at the previous bar.

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the indicator is lower than the previous one, and the corresponding price peak is higher than the previous one. In addition, the indicator must not fall below the zero level. 
No objections to buying	The indicator value grows at the analyzed bar.
No objections to selling	The indicator value falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

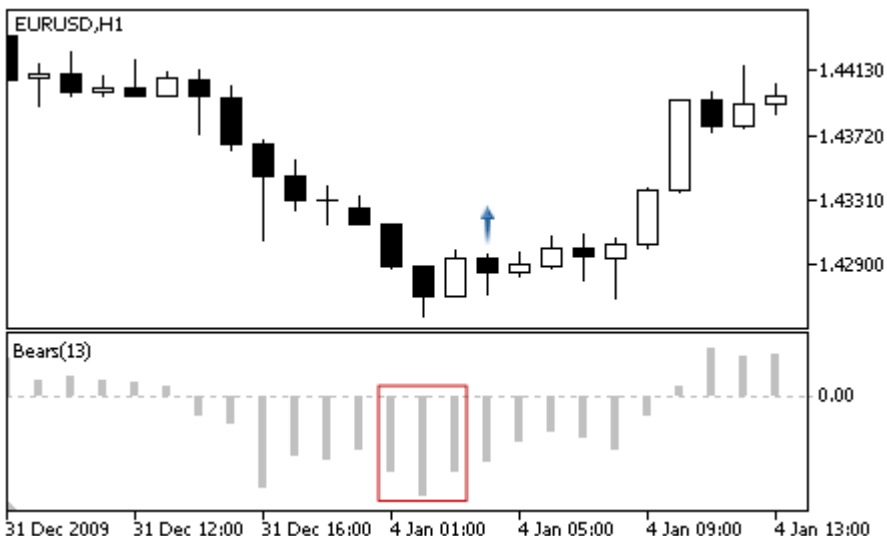
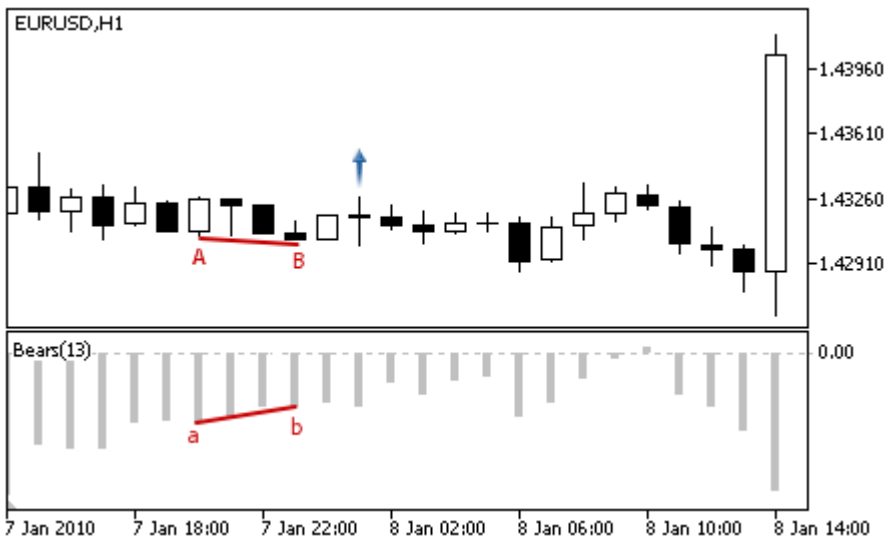
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

Signals of the Oscillator Bears Power

This module of signals is based on the market models of the oscillator [Bears Power](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse — the oscillator turned upwards and its value at the analyzed bar is below 0.  Divergence — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. In addition, the oscillator must not rise above the zero level. 
For selling	No signals for selling.

Signal Type	Description of Conditions
No objections to buying	Value of the oscillator is less than 0.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

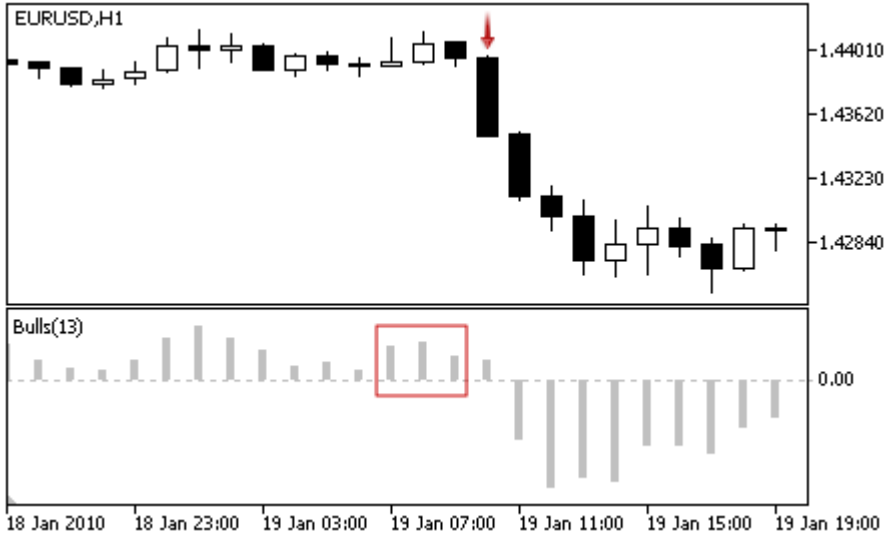
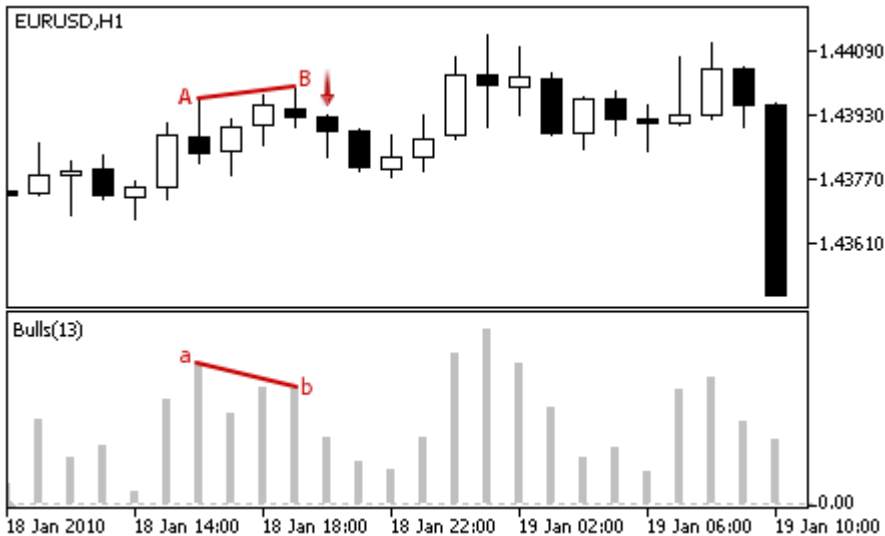
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBears	Period of calculation of the oscillator.

Signals of the Oscillator Bulls Power

This module of signals is based on the market models of the oscillator [Bulls Power](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	No signals for buying.
For selling	<ul style="list-style-type: none"> Reverse – the oscillator turned downwards and its value at the analyzed bar is above 0.  Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. In addition, the oscillator must not fall below the zero level. 

Signal Type	Description of Conditions
No objections to buying	No signals.
No objections to selling	Value of the oscillator is greater than 0.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBulls	Period of calculation of the oscillator.

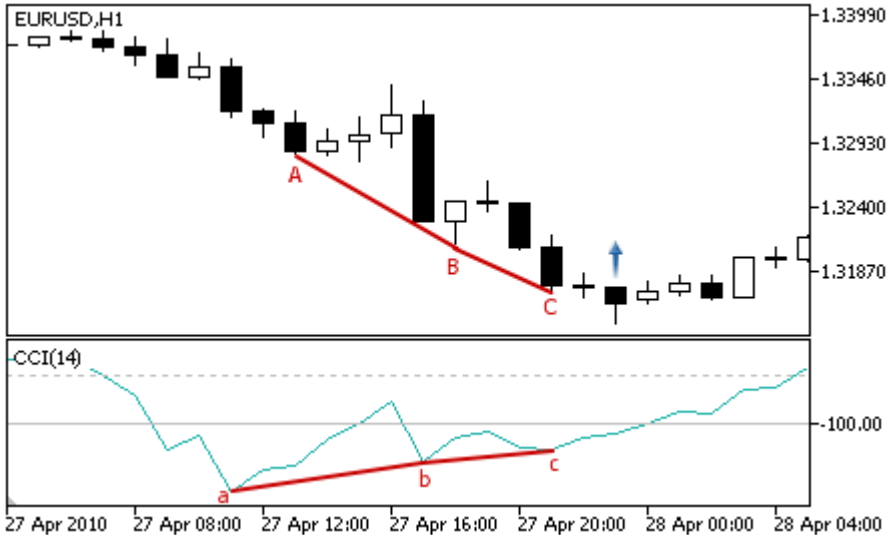
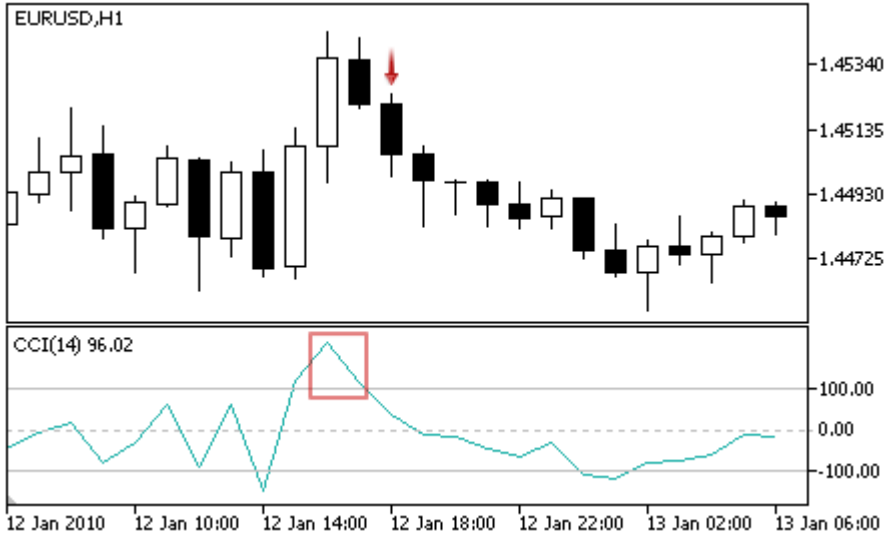
Signals of the Oscillator Commodity Channel Index

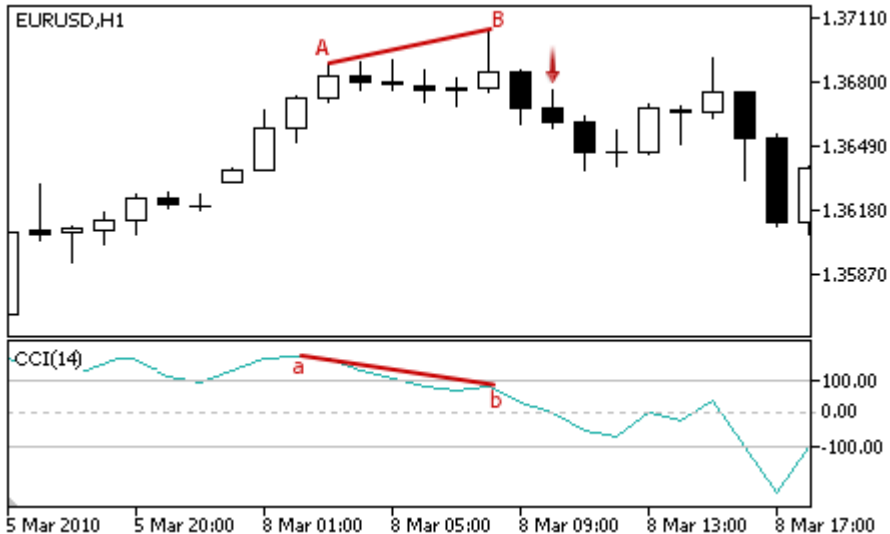

This module of signals is based on the market models of the oscillator [Commodity Channel Index](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse behind the oversold level – the oscillator turned upwards and its value at the analyzed bar is behind the level of oversold level (default value is -100).  Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. 
For selling	<ul style="list-style-type: none"> • Reverse behind the overbought level – the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 100).  <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

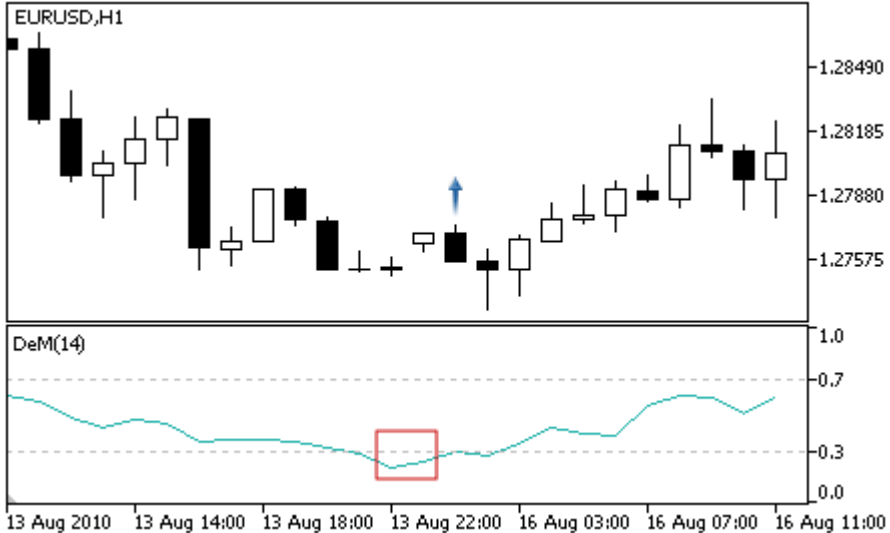
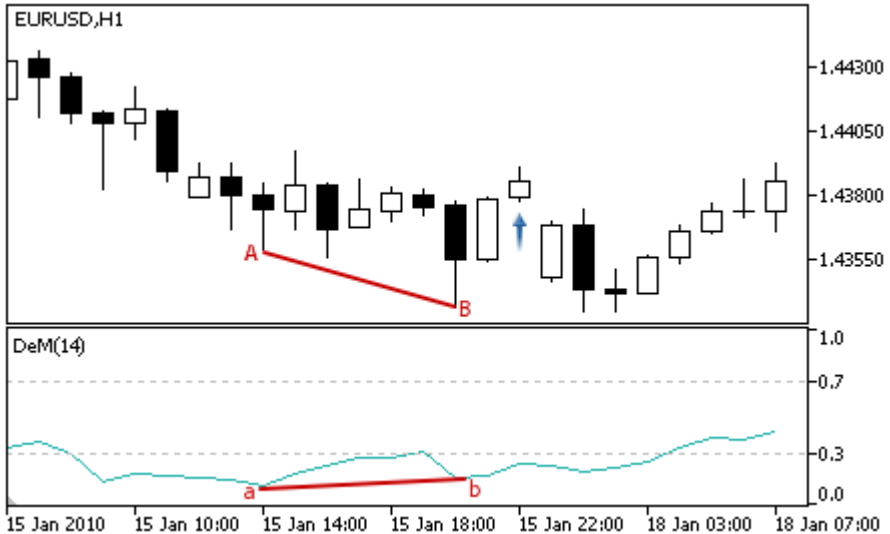
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodCCI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

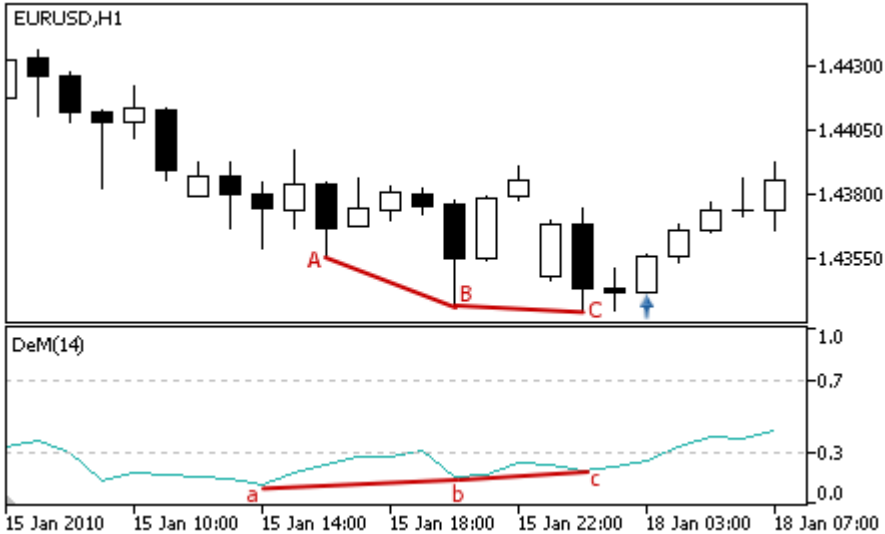
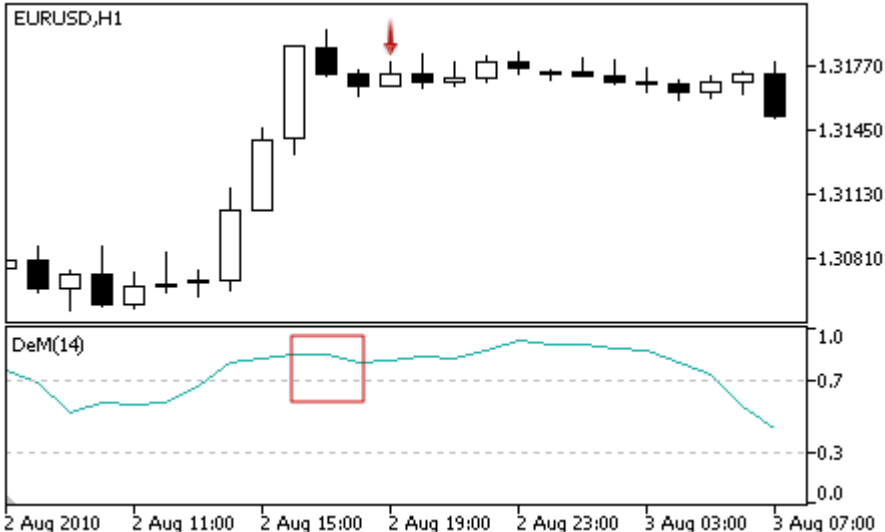
Signals of the Oscillator DeMarker

This module of signals is based on the market models of the oscillator [DeMarker](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse behind the oversold level – the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is 0.3).  Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.  Double divergence – the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. 

Signal Type	Description of Conditions
	 <p>The top chart is a candlestick plot for EURUSD on an H1 timeframe. It shows a series of price bars from January 15, 2010, to January 18, 2010. A red line connects three points labeled A, B, and C, indicating a downward trend. The bottom chart is the DeM(14) oscillator, showing a green line with points a, b, and c marked, corresponding to the price points above. The oscillator line is below the 0.7 level.</p>
For selling	<ul style="list-style-type: none"> • Reverse behind the overbought level – the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 0.7).  <p>The top chart is a candlestick plot for EURUSD on an H1 timeframe. It shows a series of price bars from August 2, 2010, to August 3, 2010. A red arrow points to a bar where the price starts to decline after an upward trend. The bottom chart is the DeM(14) oscillator, showing a green line with a peak highlighted by a red box. The oscillator line is above the 0.7 level.</p> <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 <p>EURUSD,H1</p> <p>DeM(14)</p> <ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.  <p>EURUSD,H1</p> <p>DeM(14)</p>
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

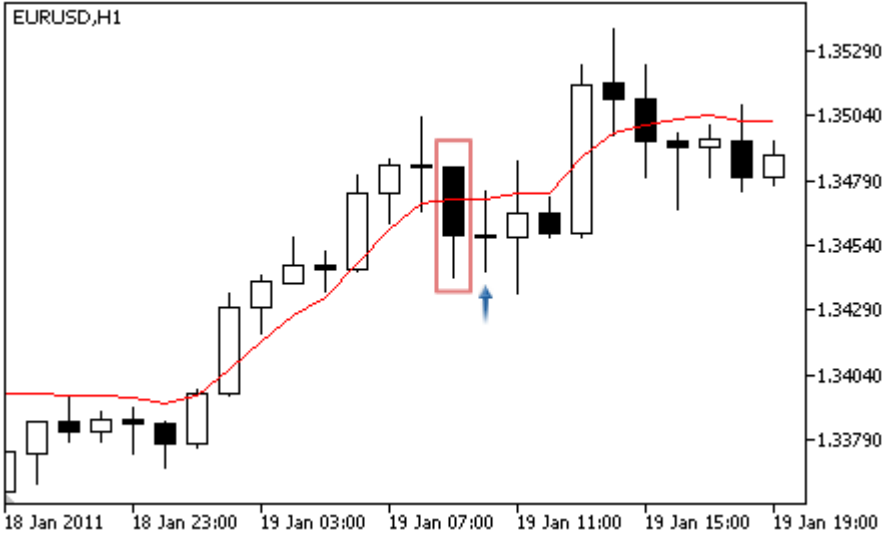
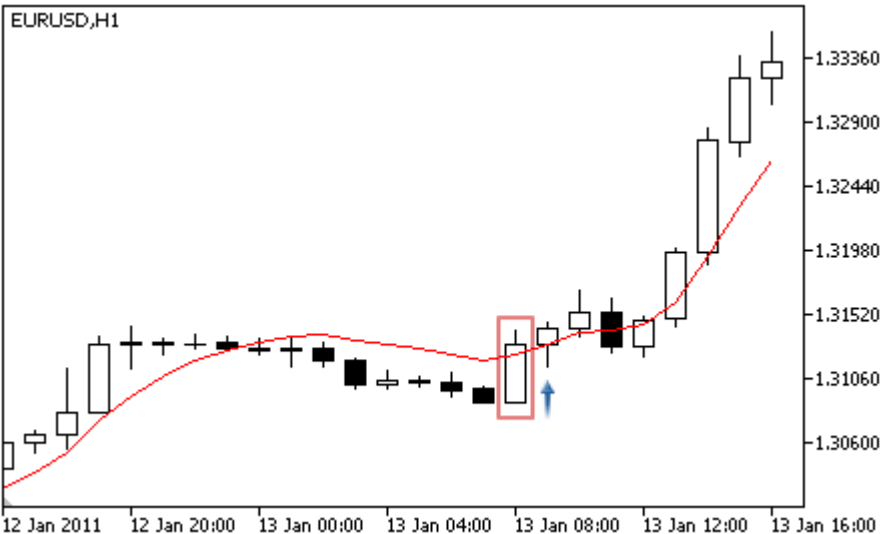
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodDeM	Period of calculation of the oscillator.

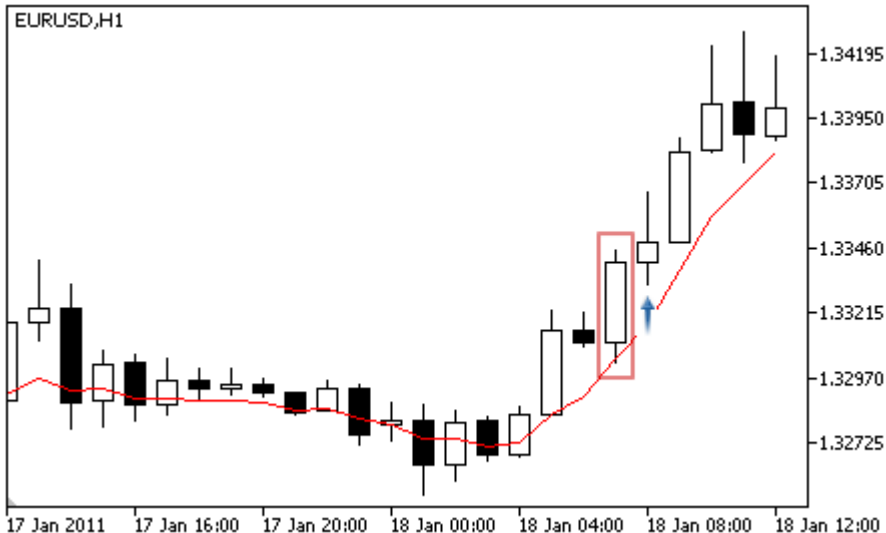
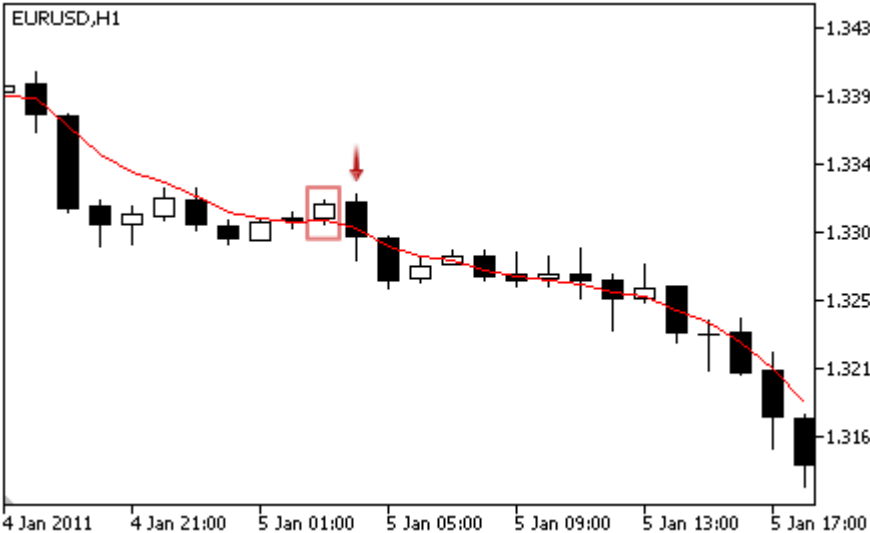
Signals of the Indicator Double Exponential Moving Average

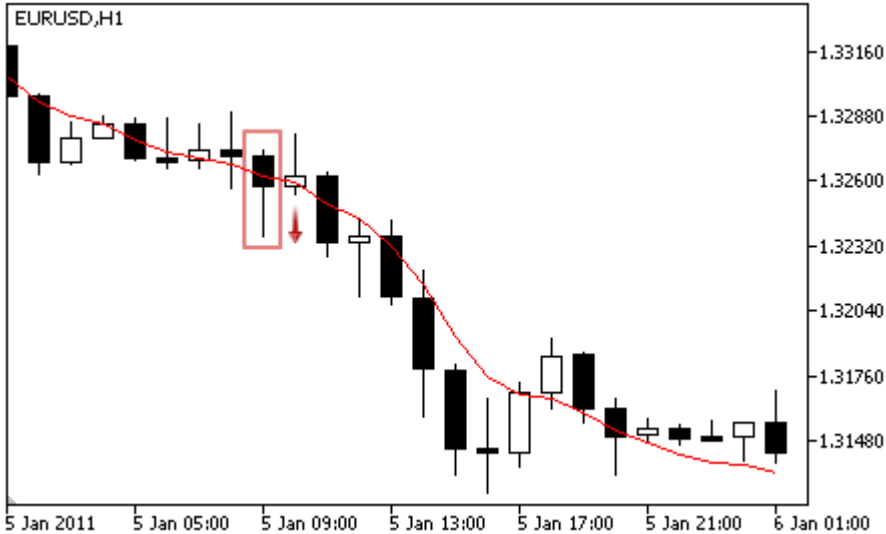
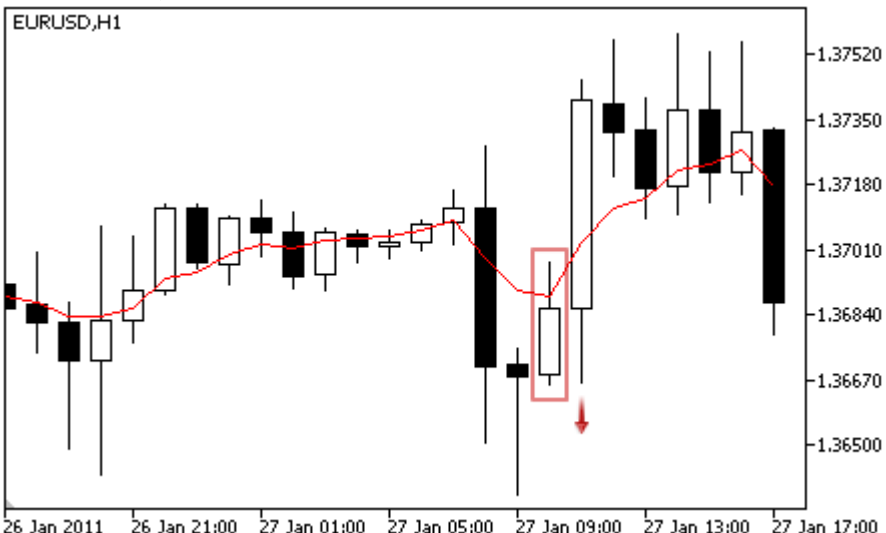
This module is based on the market models of the indicator [Double Exponential Moving Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Failed breakout. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).  Moving Average crossover. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Formed breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (signal for a roll-back from the indicator line).  <p>The chart displays EURUSD on an H1 timeframe. A red moving average line is shown. A candlestick is highlighted with a red box, indicating a breakout where the lower shadow crosses the indicator. The indicator line is shown rising, signaling a potential roll-back.</p>
For selling	<ul style="list-style-type: none"> • Failed breakout. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).  <p>The chart displays EURUSD on an H1 timeframe. A red moving average line is shown. A candlestick is highlighted with a red box, indicating a failed breakout where the price crosses the indicator upwards. The indicator line is shown falling, signaling a potential roll-back.</p> <ul style="list-style-type: none"> • Moving Average crossover. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 <p>• Formed breakout. The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).</p> 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:


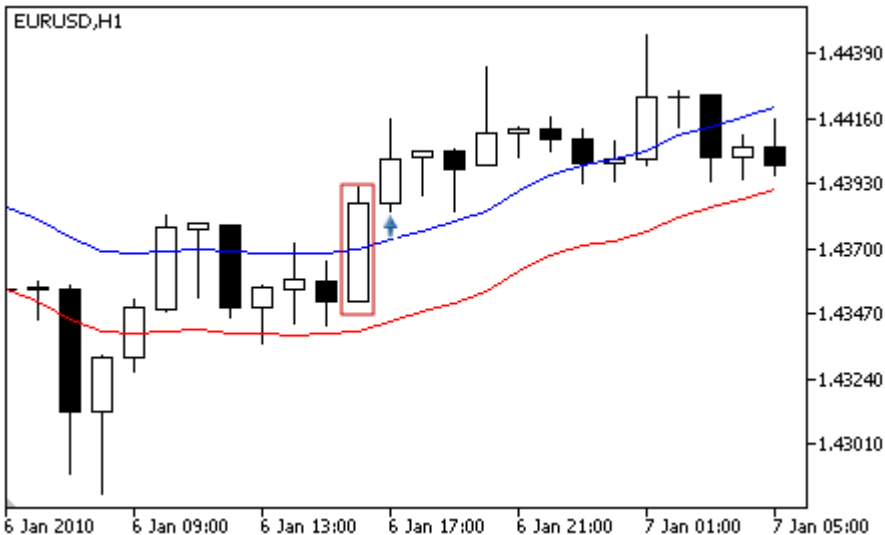
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Indicator Envelopes

This module of signals is based on the market models of the indicator [Envelopes](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> The price is near the lower line of the indicator at the analyzed bar.  <ul style="list-style-type: none"> The price crossed the upper line of the indicator at the analyzed bar. 
For selling	<ul style="list-style-type: none"> The price is near the upper line of the indicator at the analyzed bar.

Signal Type	Description of Conditions
	 <p>EURUSD, H1</p> <ul style="list-style-type: none"> The price crossed the lower line of the indicator at the analyzed bar.  <p>EURUSD, H1</p>
No objections to buying	No signals.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of calculation of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.
Deviation	Deviation of the envelope borders from the center line (MA) in percentage terms.

Signals of the Indicator Fractal Adaptive Moving Average

This module of signals is based on the market models of the indicator [Fractal Adaptive Moving Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Failed breakout. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).  Moving Average crossover. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Formed breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (signal for a roll-back from the indicator line). 
For selling	<ul style="list-style-type: none"> • Failed breakout. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).  <ul style="list-style-type: none"> • Moving Average crossover. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 <p>• Formed breakout. The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).</p> 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Intraday Time Filter

This module is based on the assumption that the efficiency of market models changes in time. Using this module, you can filter signals received from the other modules by hour and days of week. It allows increasing the quality of generated signals due to cutting off the unfavorable time periods. The mechanism of making trade decisions on the basis of signals of the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	No signals.
For selling	No signals.
No objections to buying	The current date and time meet the specified parameters.
No objections to selling	The current date and time meet the specified parameters.

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
GoodHourOfDay	Number of the only hour of day (from 0 to 23) when trade signals will be enabled. If the value is -1, the signals will be enabled through the whole day.
BadHoursOfDay	The bit field. Each bit of this field corresponds to an hour of day (0 bit - 0 hour, ..., 23 bit - 23-rd hour). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding hour. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding hour. A specified number is represented as a binary number and is used as bit mask. Disabled hours have higher priority than the enabled ones.
GoodDayOfWeek	Number of the only day of week (from 0 to 6, where 0 is Sunday), when trade signals will be

Parameter	Description
	enabled. If the value is -1, the signals will be enabled on any day.
BadDaysOfWeek	<p>The bit field. Each bit of this field corresponds to a day of week (0 bit - Sunday, ..., 6 bit - Saturday). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding day. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding day. A specified number is represented as a binary number and is used as bit mask.</p> <p>Disabled days have higher priority than the enabled ones.</p>

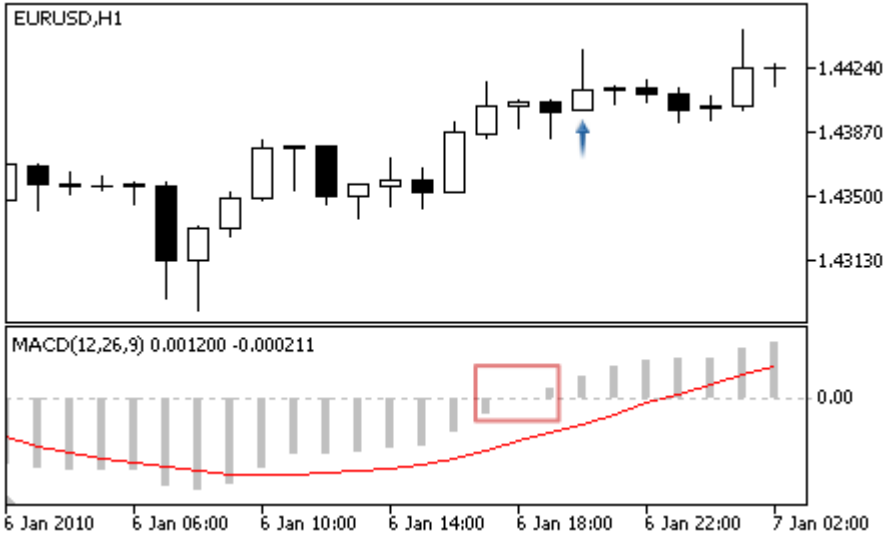
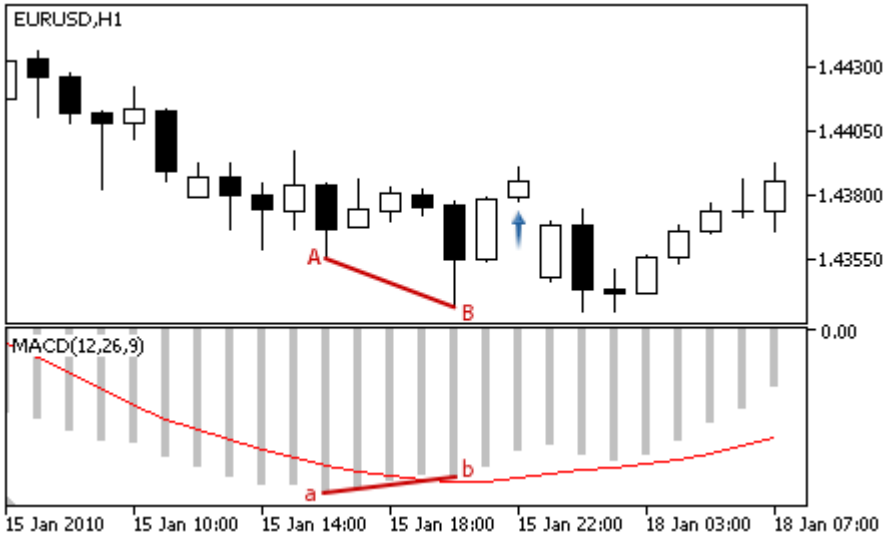
Signals of the Oscillator MACD

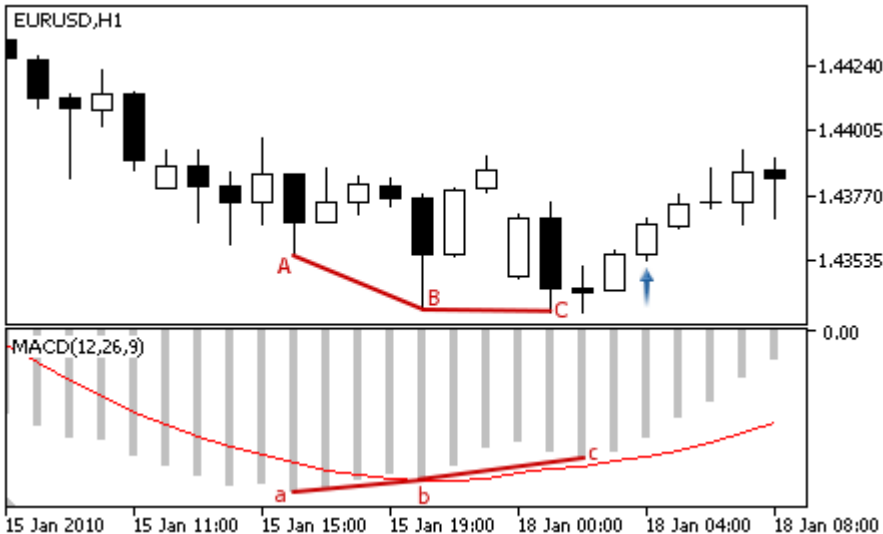
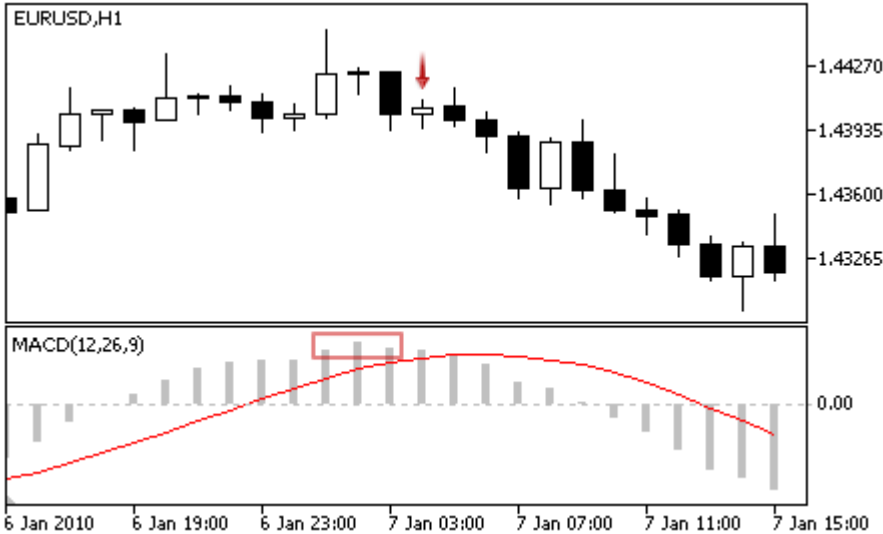
This module of signals is based on the market models of the oscillator [MACD](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse – the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  Crossover of the main and signal line – the main line is above the signal line at the analyzed bar and below the signal line at the previous one.  Crossing the zero level – the main line is above the zero level at the analyzed bar and below the zero level at the previous one. 

Signal Type	Description of Conditions
	<div data-bbox="435 315 1321 846">  <p>EURUSD,H1</p> <p>MACD(12,26,9) 0.001200 -0.000211</p> </div> <ul style="list-style-type: none"> • Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. <div data-bbox="435 958 1321 1489">  <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator forms three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.

Signal Type	Description of Conditions
	 <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>15 Jan 2010 15 Jan 11:00 15 Jan 15:00 15 Jan 19:00 18 Jan 00:00 18 Jan 04:00 18 Jan 08:00</p>
For selling	<ul style="list-style-type: none"> • Reverse – the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 19:00 6 Jan 23:00 7 Jan 03:00 7 Jan 07:00 7 Jan 11:00 7 Jan 15:00</p> <ul style="list-style-type: none"> • Crossover of the main and signal line – the main line is below the signal line at the analyzed bar and above the signal line at the previous one.

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846"> <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 19:00 6 Jan 23:00 7 Jan 03:00 7 Jan 07:00 7 Jan 11:00 7 Jan 15:00</p> </div> <ul style="list-style-type: none"> • Crossing the zero level – the main line is below the zero level at the analyzed bar and above the zero level at the previous one. <div data-bbox="432 958 1321 1489"> <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 21:00 7 Jan 01:00 7 Jan 05:00 7 Jan 09:00 7 Jan 13:00 7 Jan 17:00</p> </div> <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846"> <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>22 Jan 2010 22 Jan 05:00 22 Jan 09:00 22 Jan 13:00 22 Jan 17:00 22 Jan 21:00 25 Jan 02:00</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. <div data-bbox="432 987 1321 1518"> <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>11 Jan 2010 11 Jan 05:00 11 Jan 09:00 11 Jan 13:00 11 Jan 17:00 11 Jan 21:00 12 Jan 01:00</p> </div>
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

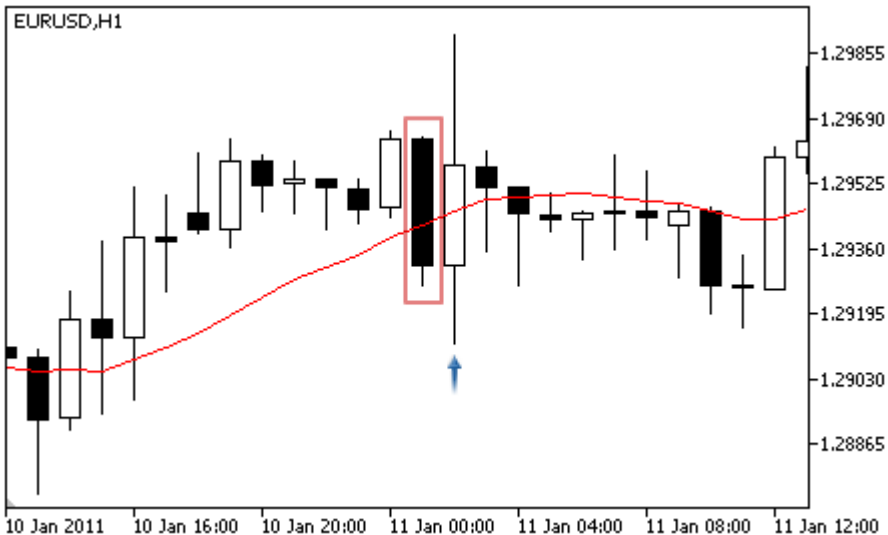
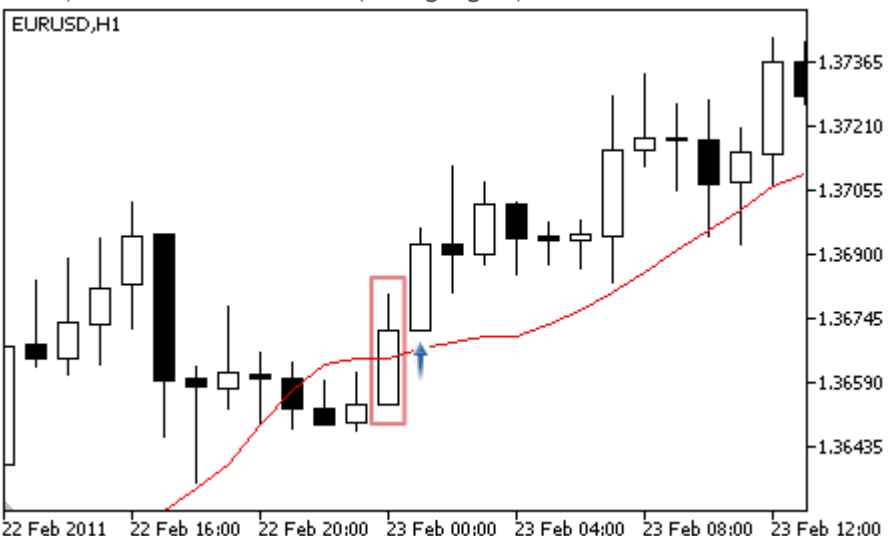
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodFast	Period of calculation of the fast EMA.
PeriodSlow	Period of calculation of the slow EMA.
PeriodSignal	Period of smoothing.
Applied	A price series used for calculation of the oscillator.

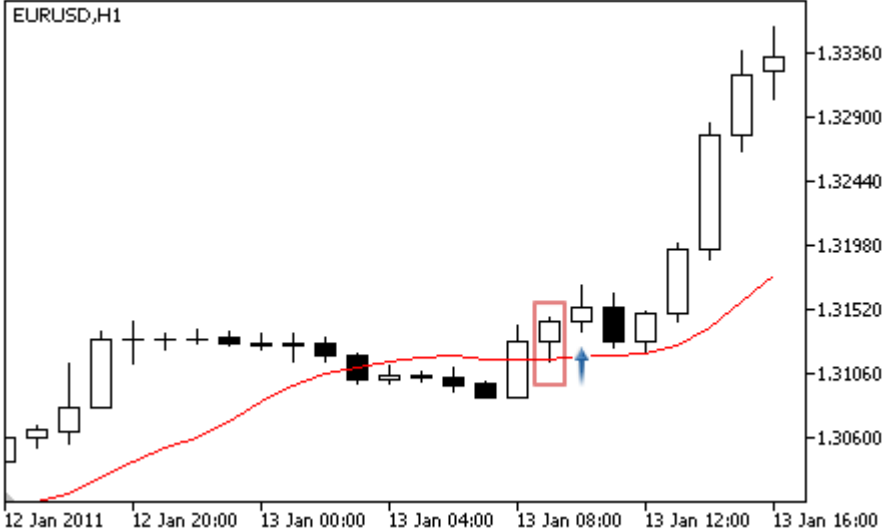
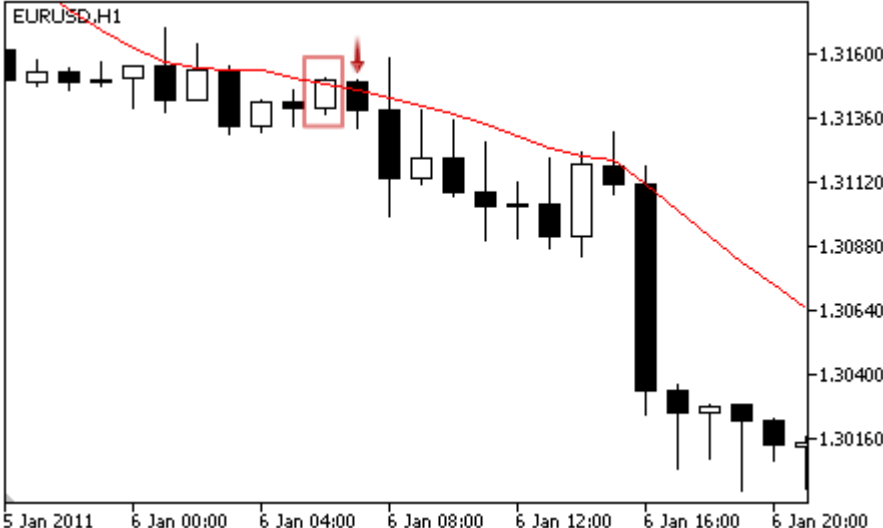
Signals of the Indicator Moving Average

This module of signals is based on the market models of the indicator [Moving Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Failed breakout. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).  Moving Average crossover. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Formed breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line). 
For selling	<ul style="list-style-type: none"> • Failed breakout. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  <ul style="list-style-type: none"> • Moving Average crossover. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> • Formed breakout. The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

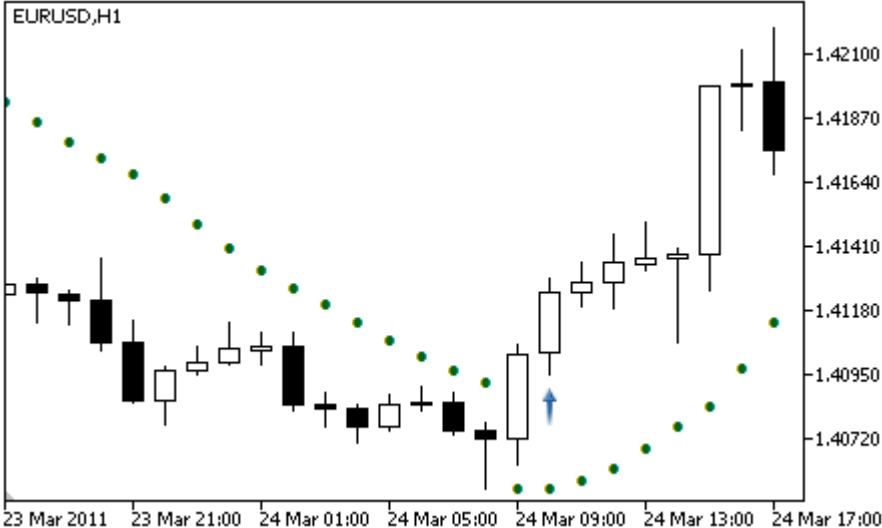
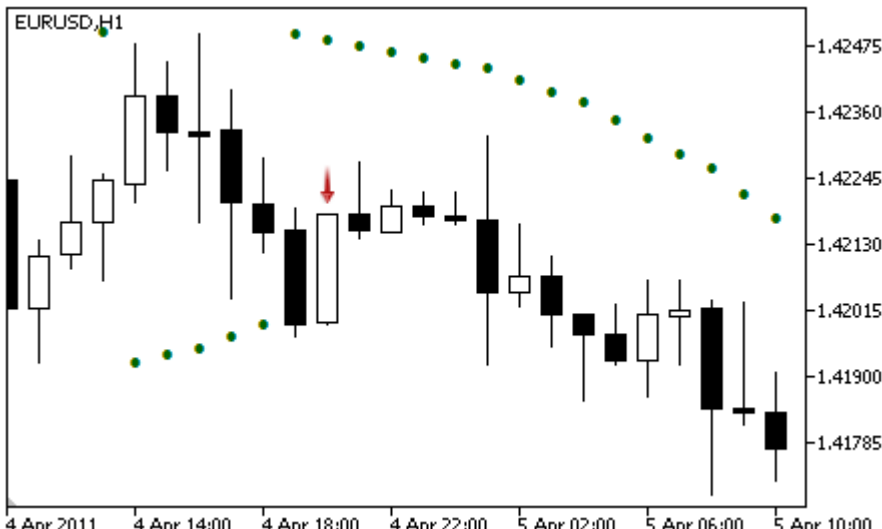
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the indicator Parabolic SAR

This module of signals is based on the market models of the indicator [Parabolic SAR](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<p>Reverse — the indicator is below the price at the analyzed bar and above the price at the previous one.</p>  <p>The chart displays EURUSD on an H1 timeframe from March 23 to March 24, 2011. The price is shown as candlesticks, and the Parabolic SAR indicator is represented by green dots. A blue arrow points upwards from the indicator to the price bar, indicating a 'Reverse' buy signal.</p>
For selling	<p>Reverse — the indicator is above the price at the analyzed bar and below the price at the previous one.</p>  <p>The chart displays EURUSD on an H1 timeframe from April 4 to April 5, 2011. The price is shown as candlesticks, and the Parabolic SAR indicator is represented by green dots. A red arrow points downwards from the indicator to the price bar, indicating a 'Reverse' sell signal.</p>
No objections	The price is above the indicator.

Signal Type	Description of Conditions
to buying	
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

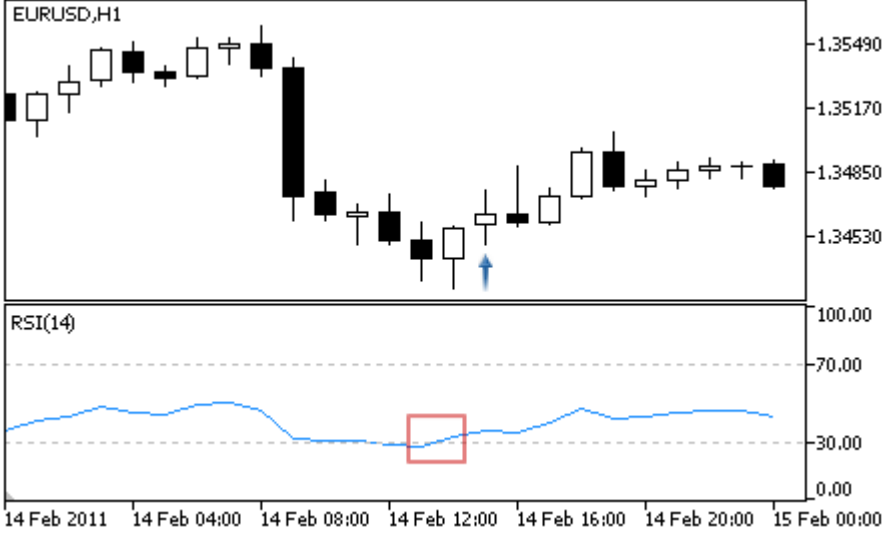
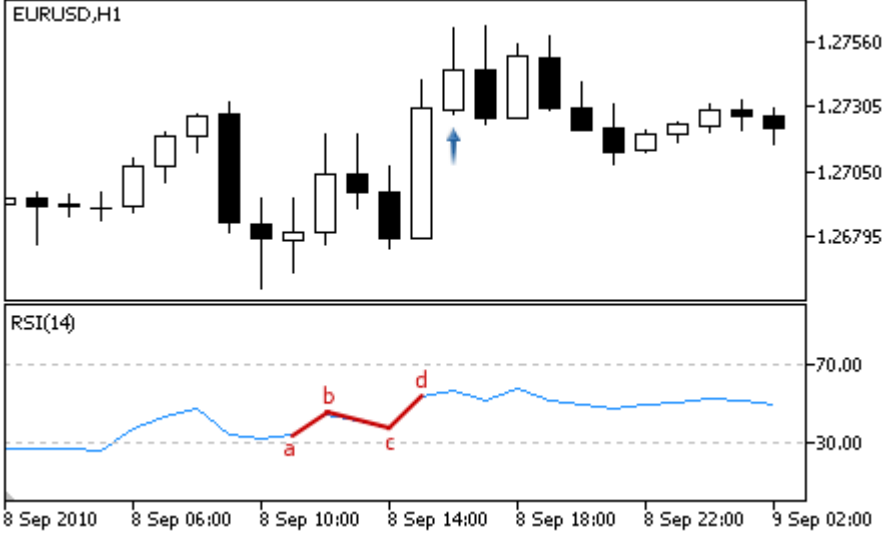
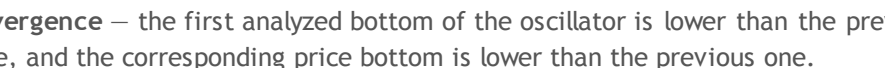
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
Step	The step of price increment.
Maximum	Maximum rate of the speed of convergence of the indicator with the price.

Signals of the Oscillator Relative Strength Index

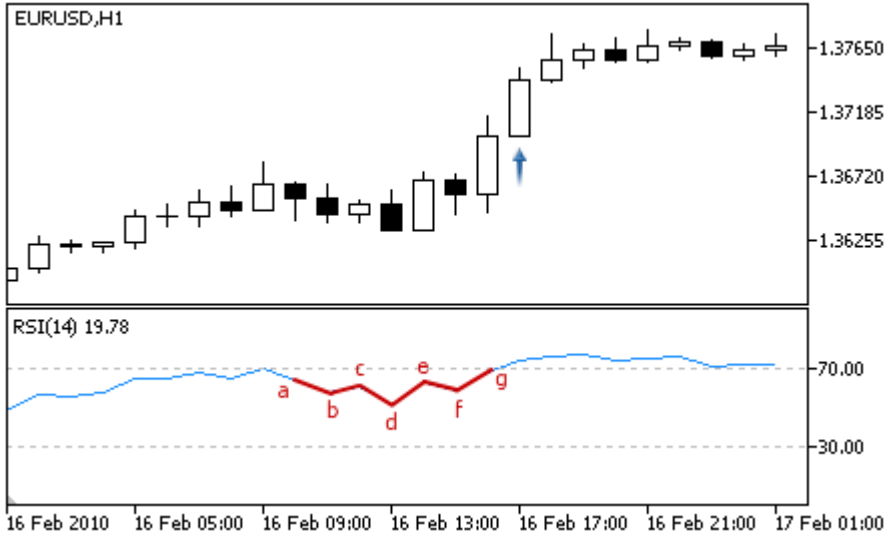
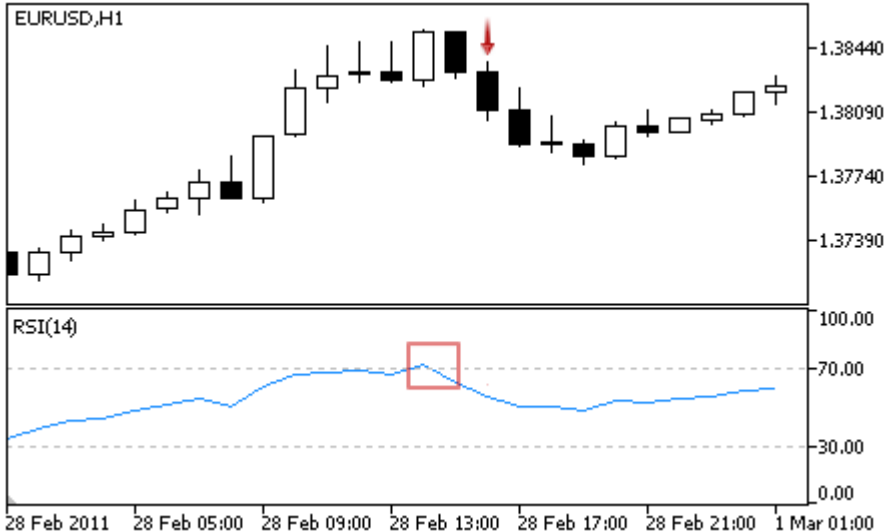
This module of signals is based on the market models of the oscillator [Relative Strength Index](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

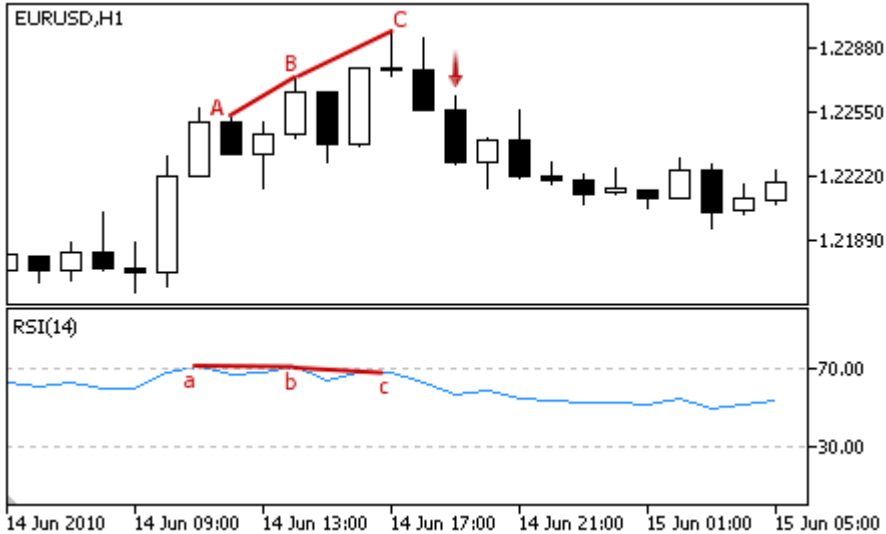
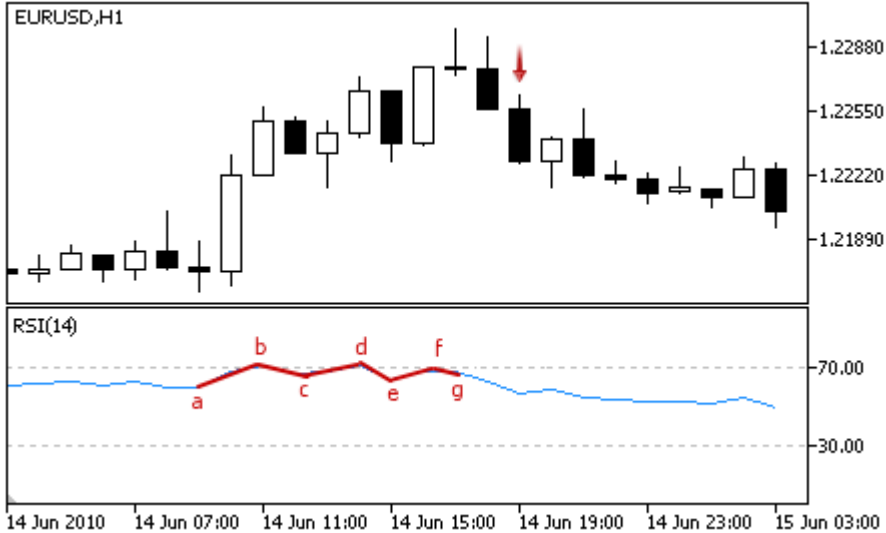
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse behind the oversold level – the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is 30).  Failed swing – the oscillator rises higher than the previous peak at the analyzed bar.  Divergence – the first analyzed bottom of the oscillator is lower than the previous one, and the corresponding price bottom is lower than the previous one. 

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846"> <p>EURUSD, H1</p> <p>RSI(14)</p> <p>20 Aug 2010 20 Aug 10:00 20 Aug 14:00 20 Aug 18:00 20 Aug 22:00 23 Aug 02:00 23 Aug 06:00</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. <div data-bbox="432 992 1321 1523"> <p>EURUSD, H1</p> <p>RSI(14)</p> <p>11 Nov 2010 11 Nov 19:00 11 Nov 23:00 12 Nov 03:00 12 Nov 07:00 12 Nov 11:00 12 Nov 15:00</p> </div> <ul style="list-style-type: none"> • Head/Shoulders – the oscillator formed three consequent bottoms, and the mid one is lower than the others.

Signal Type	Description of Conditions
	 <p>EURUSD,H1</p> <p>RSI(14) 19.78</p> <p>16 Feb 10:00 16 Feb 16:00 16 Feb 22:00 17 Feb 01:00</p>
For selling	<ul style="list-style-type: none"> • Reverse behind the overbought level – the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 70).  <p>EURUSD,H1</p> <p>RSI(14)</p> <p>28 Feb 08:00 28 Feb 14:00 28 Feb 20:00 1 Mar 01:00</p> <ul style="list-style-type: none"> • Failed swing – the oscillator falls lower than the previous bottom at the analyzed bar.

Signal Type	Description of Conditions
	<div data-bbox="430 313 1316 840"> <p>EURUSD,H1</p> <p>RSI(14)</p> <p>23 Aug 2010 23 Aug 06:00 23 Aug 10:00 23 Aug 14:00 23 Aug 18:00 23 Aug 22:00 24 Aug 02:00</p> </div> <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. <div data-bbox="430 963 1316 1489"> <p>EURUSD,H1</p> <p>RSI(14)</p> <p>21 Sep 2010 22 Sep 02:00 22 Sep 06:00 22 Sep 10:00 22 Sep 14:00 22 Sep 18:00 22 Sep 22:00</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.

Signal Type	Description of Conditions
	 <p>• Head/Shoulders — the oscillator formed three consequent peaks, and the mid one is higher than the others.</p> 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

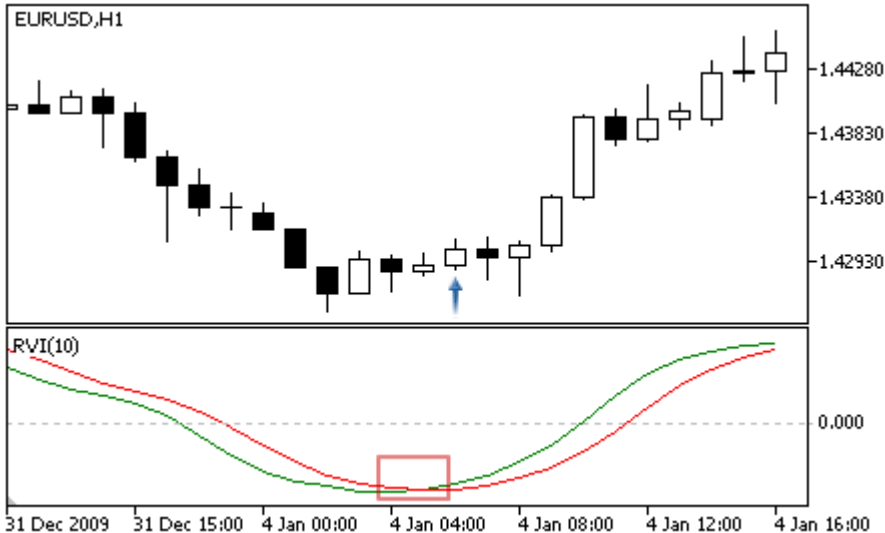
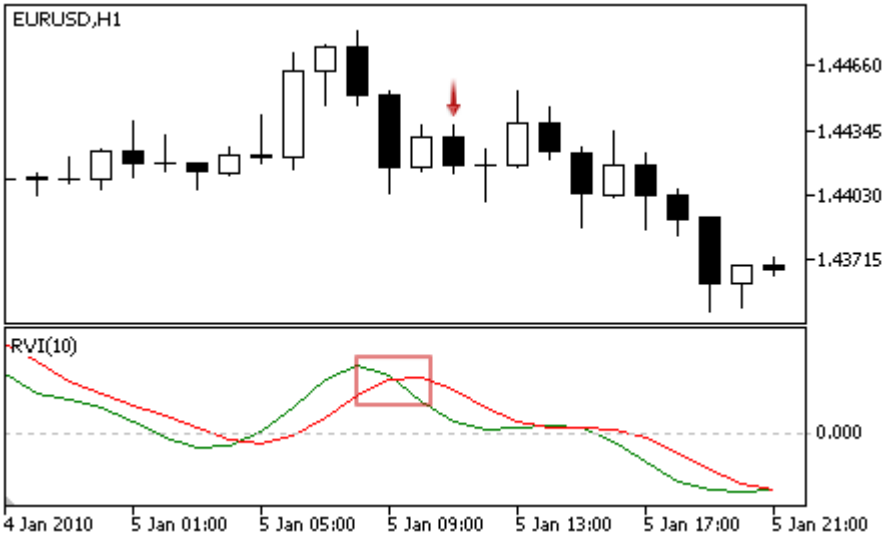
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRSI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

Signals of the Oscillator Relative Vigor Index

This module of signals is based on the market models of the oscillator [Relative Vigor Index](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<p>Crossing of the main and signal line – the main line is above the signal line at the analyzed bar and below the signal line at the previous one.</p> 
For selling	<p>Crossing of the main and signal line – the main line is below the signal line at the analyzed bar and above the signal line at the previous one.</p> 
No objections	Value of the oscillator grows at the analyzed bar.

Signal Type	Description of Conditions
to buying	
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

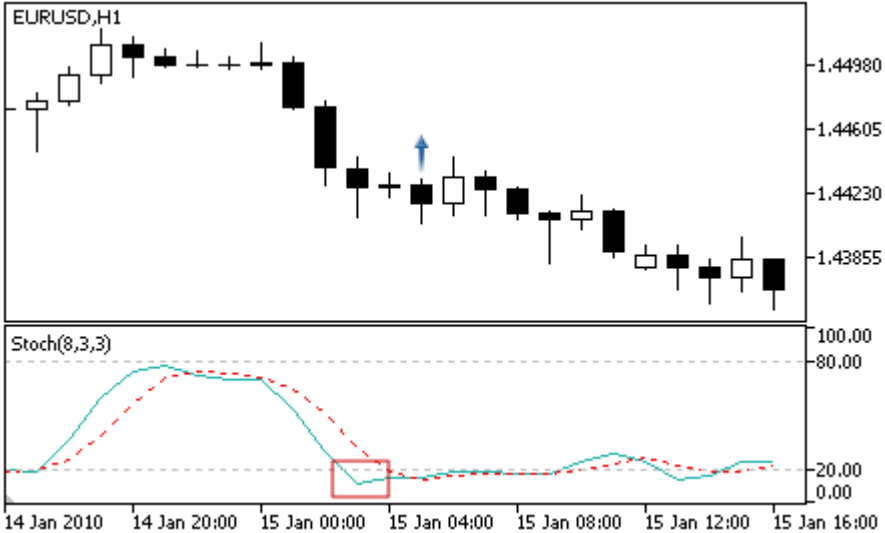
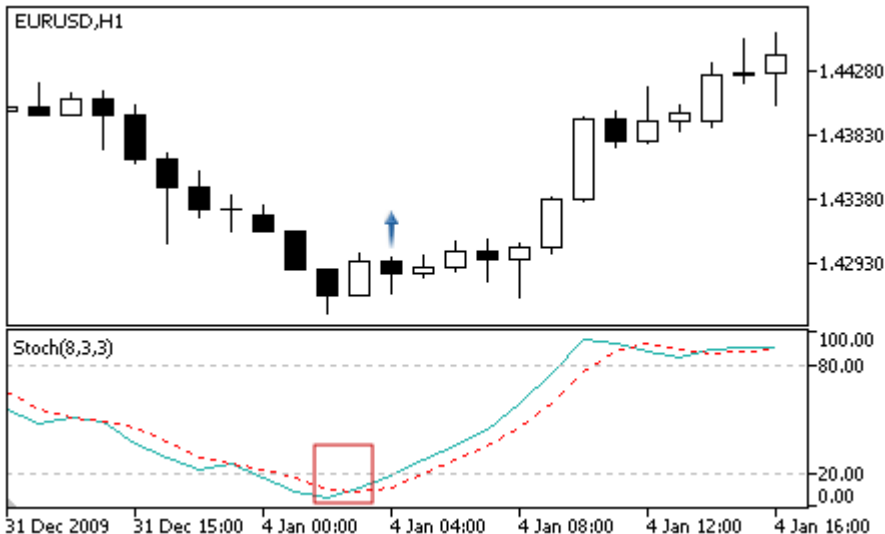
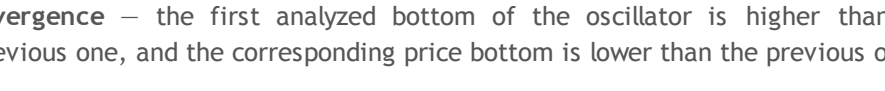
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRVI	Period of calculation of the oscillator.

Signals of the Oscillator Stochastic

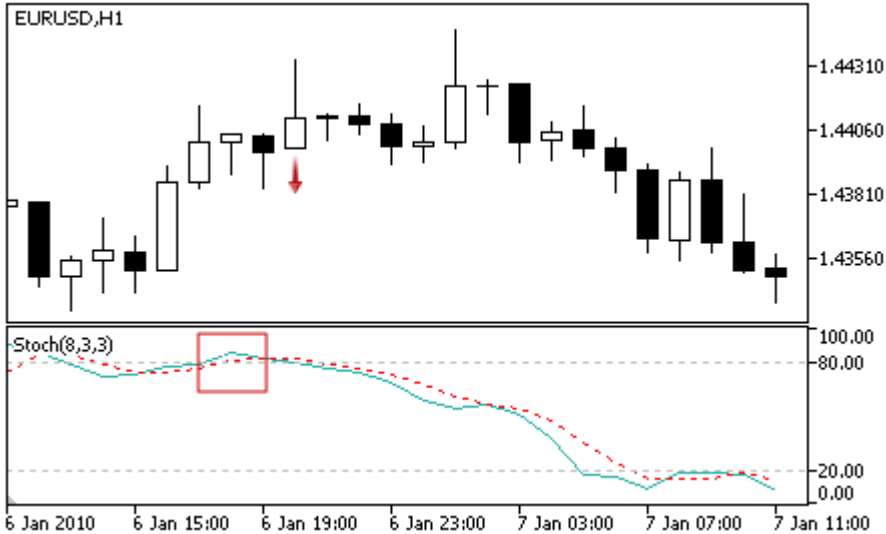
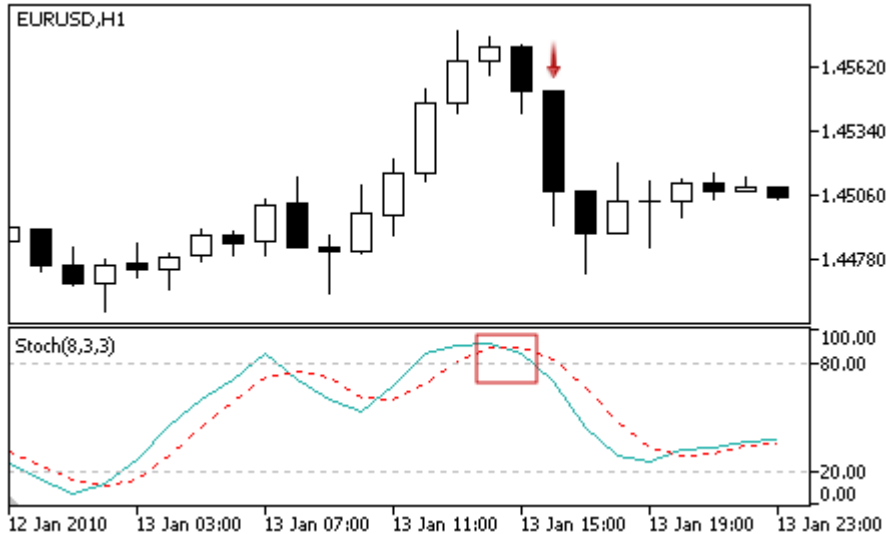
This module of signals is based on the market models of the oscillator [Stochastic](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse – the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  Crossing of the main and signal line – the main line is above the signal line at the analyzed bar and below the signal line at the previous one.  Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846"> <p>EURUSD, H1</p> <p>Stoch(8,3,3)</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. <div data-bbox="432 992 1321 1525"> <p>EURUSD, H1</p> <p>Stoch(8,3,3)</p> </div>
For selling	<ul style="list-style-type: none"> • Reverse – the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846">  <p>EURUSD,H1</p> <p>Stoch(8,3,3)</p> </div> <ul style="list-style-type: none"> • Crossing of the main and signal line – the main line is below the signal line at the analyzed bar and above the signal line at the previous one. <div data-bbox="432 958 1321 1489">  <p>EURUSD,H1</p> <p>Stoch(8,3,3)</p> </div> <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	<div data-bbox="432 315 1321 846"> <p>EURUSD,H1</p> <p>Stoch(8,3,3)</p> <p>4 Jan 2010 4 Jan 04:00 4 Jan 08:00 4 Jan 12:00 4 Jan 16:00 4 Jan 20:00 5 Jan 00:00</p> </div> <ul style="list-style-type: none"> • Double divergence – the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. <div data-bbox="432 994 1321 1525"> <p>EURUSD,H1</p> <p>Stoch(8,3,3)</p> <p>1 Feb 2010 2 Feb 00:00 2 Feb 04:00 2 Feb 08:00 2 Feb 12:00 2 Feb 16:00 2 Feb 20:00</p> </div>
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodK	Period of calculation of the main line of the oscillator.
PeriodD	Period of averaging of the main line of the oscillator.
PeriodSlow	Period of slowing.
Applied	A price series used for calculation of the oscillator.

Signals of the Oscillator Triple Exponential Average

This module of signals is based on the market models of the oscillator [Triple Exponential Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse – the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  Crossing the zero level – the main line is above the zero level at the analyzed bar and below the zero level at the previous one. 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 
For selling	<ul style="list-style-type: none"> • Reverse – the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  <ul style="list-style-type: none"> • Crossing the zero level – the main line is below the zero level at the analyzed bar and above the zero level at the previous one.

Signal Type	Description of Conditions
	 <ul style="list-style-type: none"> • Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

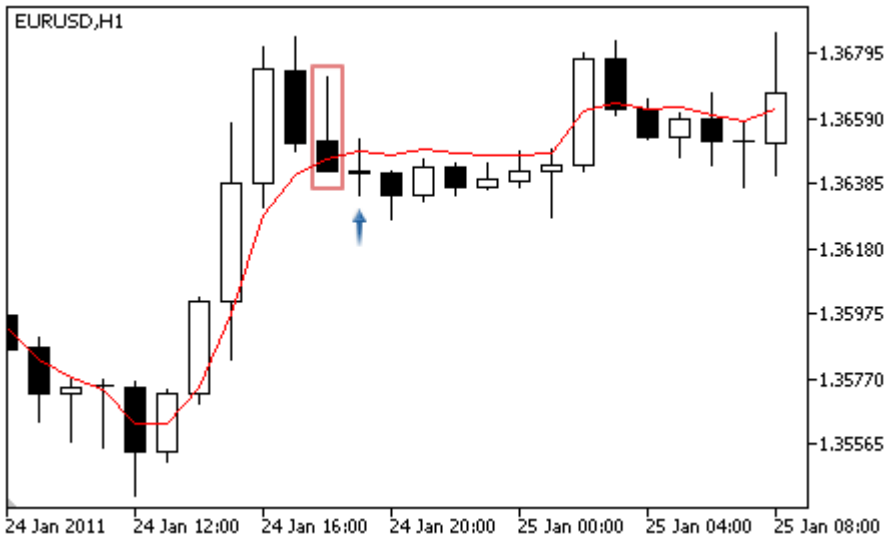
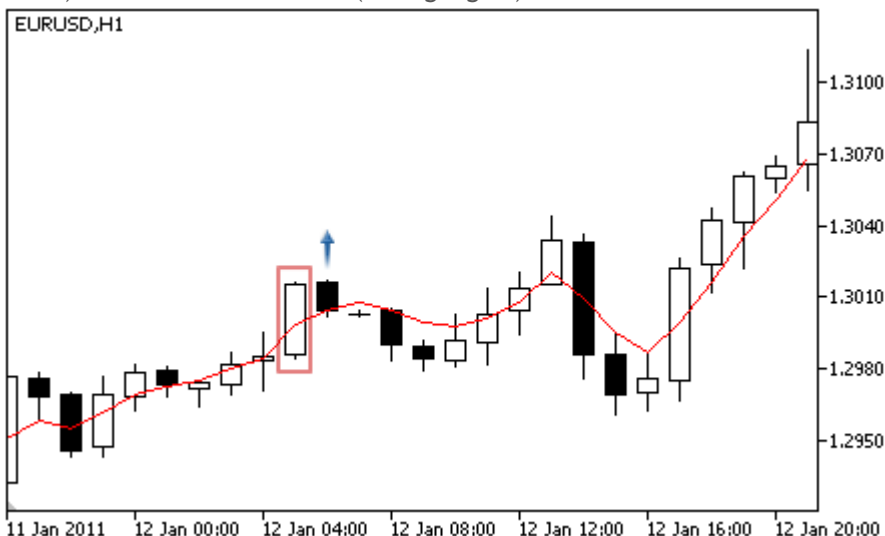
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodTriX	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

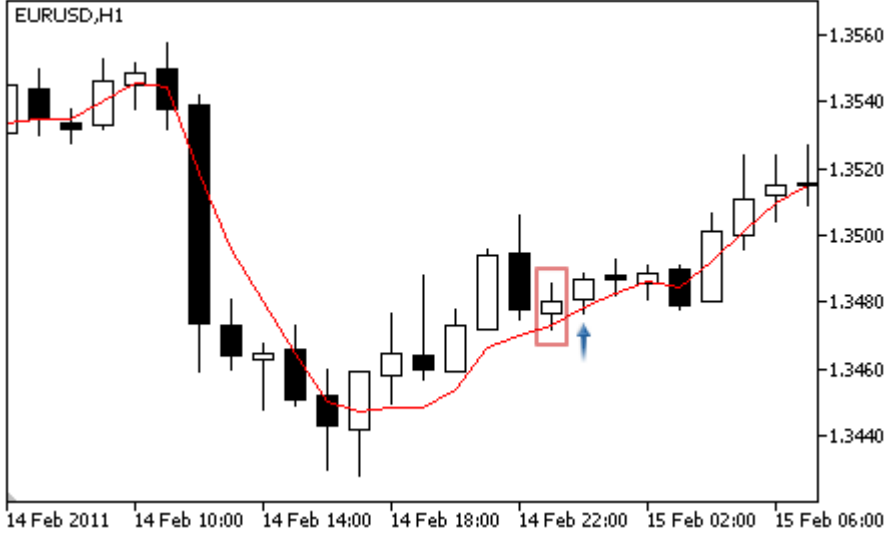
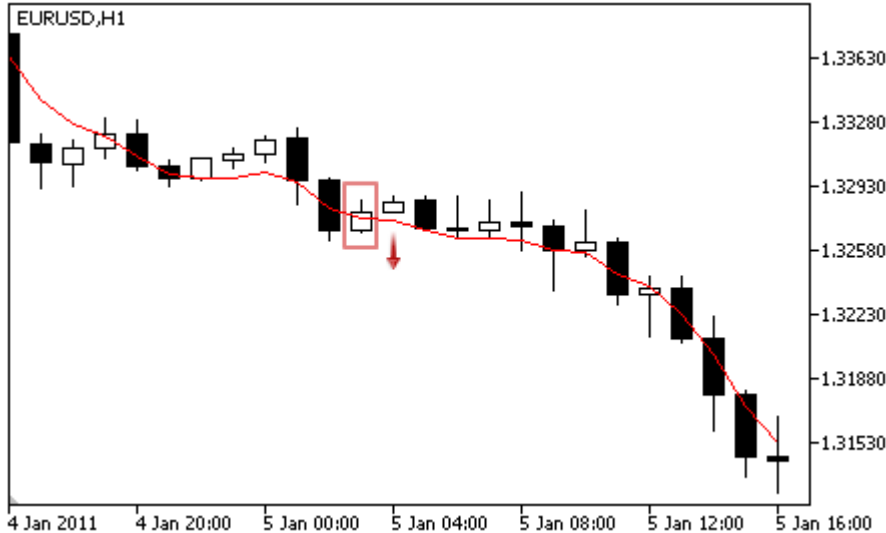
Signals of the Indicator Triple Exponential Moving Average

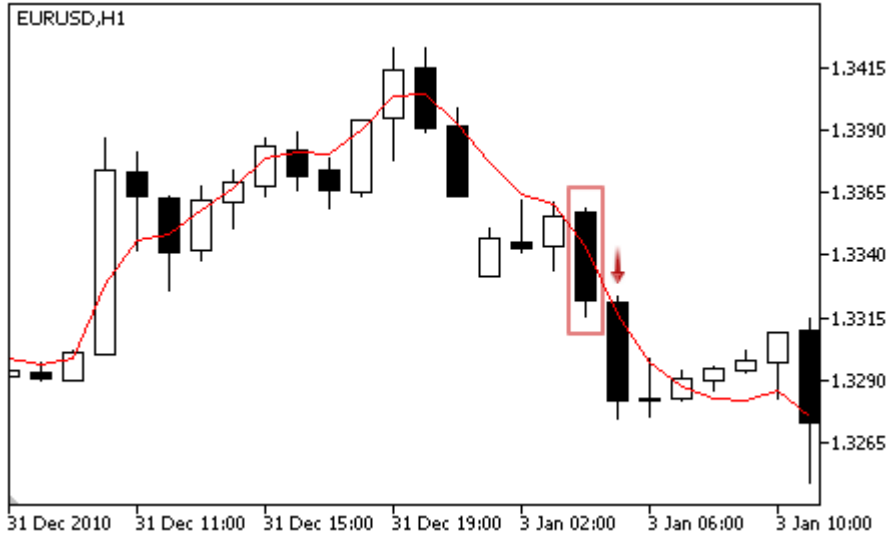
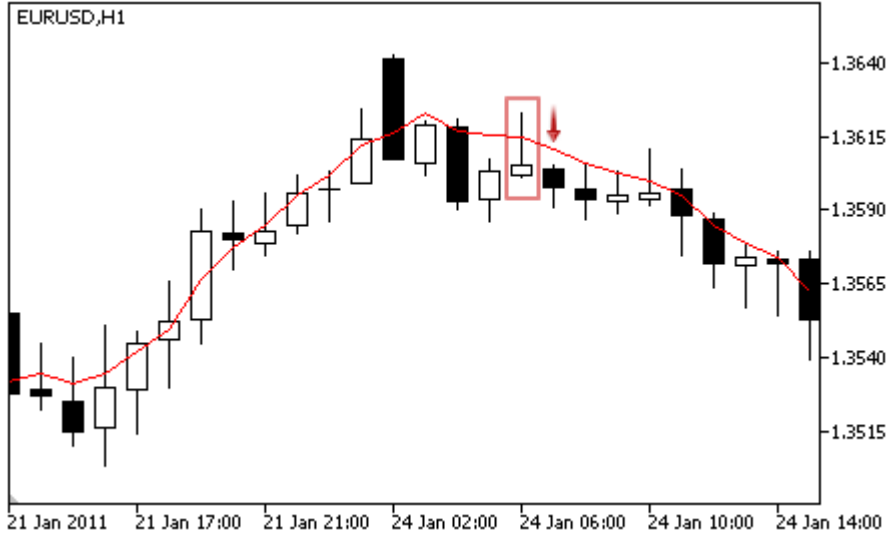
This module of signals is based on the market models of the indicator [Triple Exponential Moving Average](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal of a roll-back from the indicator line).  <ul style="list-style-type: none"> Moving Average crossover. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	<ul style="list-style-type: none"> • Formed breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal of a roll-back from the indicator line).  <p>The chart displays EURUSD on an H1 timeframe from 14 Feb 2011 to 15 Feb 2011. A red line indicator is plotted. A candlestick bar on Feb 14 at 22:00 is highlighted with a red box, showing its lower shadow crossing the indicator line. A blue arrow points to the indicator line at approximately 1.3475.</p>
For selling	<ul style="list-style-type: none"> • Failed breakout. The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal of a roll-back from the indicator line).  <p>The chart displays EURUSD on an H1 timeframe from 4 Jan 2011 to 5 Jan 2011. A red line indicator is plotted. A candlestick bar on Jan 5 at 00:00 is highlighted with a red box, showing its open price below the indicator and its close price above it. A red arrow points to the indicator line at approximately 1.32580.</p> <ul style="list-style-type: none"> • Moving Average crossover. The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 <p>• Formed breakout. The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal of a roll-back from the indicator line).</p> 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

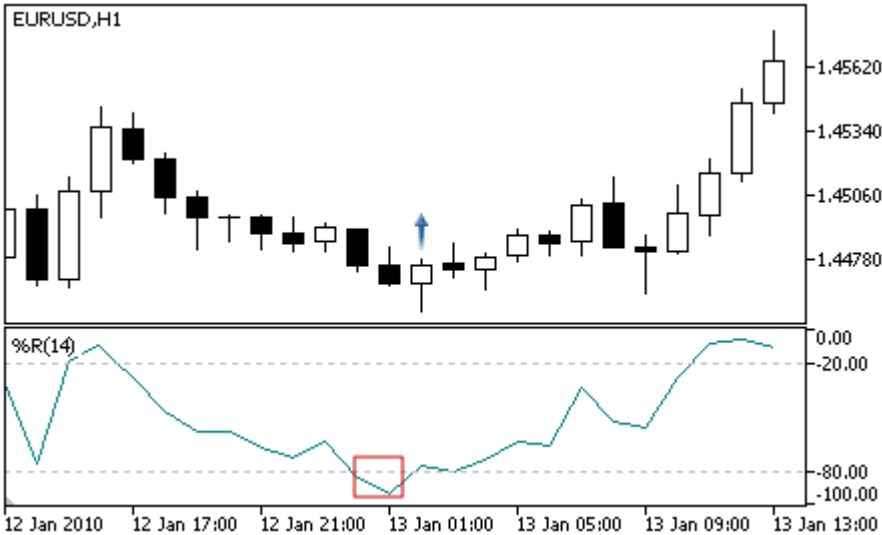
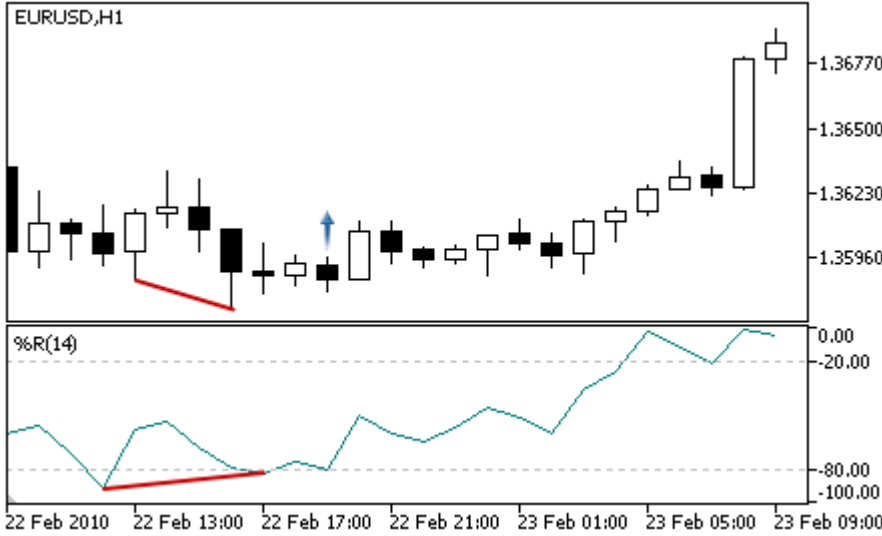
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

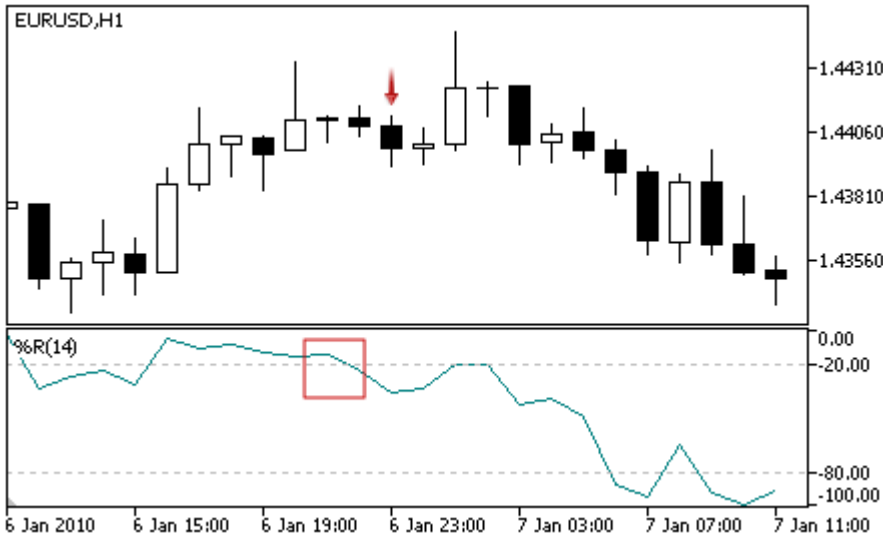
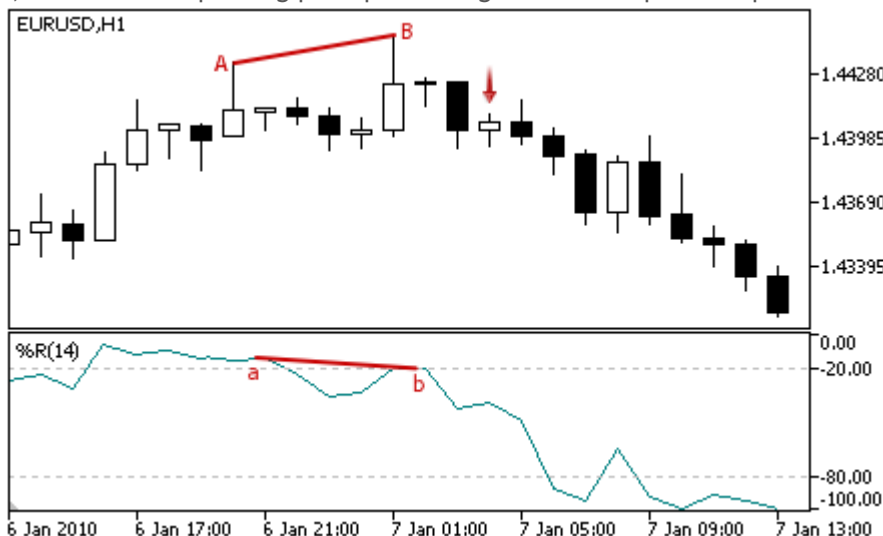
Signals of the Oscillator Williams Percent Range

This module of signals is based on the market models of the oscillator [Williams Percent Range](#). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section](#).

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	<ul style="list-style-type: none"> Reverse behind the oversold level – the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is -80).  Divergence – the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 

Signal Type	Description of Conditions
For selling	<ul style="list-style-type: none"> Reverse behind the overbought level – the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is -20).  <ul style="list-style-type: none"> Divergence – the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Remember that the oscillator Williams Percent Range has a reversed scale. Its maximum value is -100, and minimum is 0.

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodWPR	Period of calculation of the oscillator.

Trailing Stop classes

This section contains technical details of working with trailing stop classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert\Trailing folder.

Class	Description
<u>CTrailingFixedPips</u>	This class implements Trailing Stop algorithm based on fixed points
<u>CTrailingMA</u>	This class implements Trailing Stop algorithm based on the values of Moving Average indicator
<u>CTrailingNone</u>	A stub class, it does not uses any Trailing Stop algorithm
<u>CTrailingPSAR</u>	This class implements Trailing Stop algorithm based on the values of Parabolic SAR indicator

CTrailingFixedPips

CTrailingFixedPips is a class with implementation of Trailing Stop algorithm based on fixed points trailing.

CTrailingFixedPips class implements the following algorithm of trailing open positions: If Stop Loss level is equal to zero, Stop Loss order modification condition is considered unfulfilled, therefore there is no suggestion to change a position's Stop Loss. Otherwise, the check of whether the price has moved in favorable direction is performed.

If a position has a Stop Loss order, it checks the minimal allowed Stop Loss distance to the current price. If the position has no Stop Loss level, the distance between the current and open prices is checked. If the distance exceeds Stop Loss level, the system suggests to set a new Stop Loss price.

If Stop Loss modification condition is fulfilled and Take Profit is not equal to zero, the system suggests setting a new Take Profit price.

If Expert Advisor class has been [initialized](#) with the flag every_tick=false, it will perform all operations (trading, trailing, etc) only at the new bar on a working symbol and timeframe. In this case, setting Take Profit order allows you to close position when the price moves in the position direction before a new bar appears.

Description

CTrailingFixedPips implements the Trailing Stop algorithm at a specified "distance" from the current price (in points).

Declaration

```
class CTrailingFixedPips: public CExpertTrailing
```

Title

```
#include <Expert\Trailing\CTrailingFixedPips.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertTrailing](#)

CTrailingFixedPips

Class Methods by Groups

Initialization	
StopLevel	Sets the value of Stop Loss level
ProfitLevel	Sets the value of Take Profit level
virtual ValidationSettings	Checks the settings
Check Trailing Methods	

virtual CheckTrailingStopLong	Check Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

StopLevel

Sets the value of Stop Loss level.

```
void StopLevel(  
    int stop_level    // level  
)
```

Parameters

stop_loss

[in] The value of Stop Loss level (in conventional 2/4-digit points).

Note

If Stop Loss level is equal to 0, the Trailing Stop is not used.

ProfitLevel

Sets the value of Take Profit level.

```
void ProfitLevel(  
    int profit_level    // level  
)
```

Parameters

profit_level

[in] The value of Take Profit level (in conventional 2/4-digit points).

Note

If profit level is equal to 0, the Trailing Stop is not used.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function checks Take Profit and Stop Loss levels. The correct values are 0 and values greater than the minimal indention in points from the current close price to place Stop orders.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool CheckTrailingStopLong(  
    CPositionInfo* position,    // pointer  
    double& sl,                // reference to Stop Loss  
    double& tp                 // reference to Take Profit  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

If Stop Loss level is equal to 0, the condition is not fulfilled (exit). If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the current Bid price is higher than base price+stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Bid price+take profit level.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool CheckTrailingStopShort (
    CPositionInfo* position,      // pointer
    double& sl,                  // reference to Stop Loss
    double& tp                    // reference to Take Profit
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

If Stop Loss level is equal to 0, the condition is not fulfilled (exit). If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the current Ask price is lower than base price - stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Ask price - take profit level.

CTrailingMA

CTrailingMA is a class with implementation of Trailing Stop algorithm based on the values of moving average indicator.

Description

CTrailingMA class implements Trailing Stop algorithm based on the values of moving average indicator of the previous (completed) bar.

Declaration

```
class CTrailingMA: public CExpertTrailing
```

Title

```
#include <Expert\Trailing\TrailingMA.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertTrailing](#)

CTrailingMA

Class Methods by Groups

Initialization	
Period	Sets period of moving average
Shift	Sets shift of moving average
Method	Sets smoothing method of moving average
Applied	Sets applied price of moving average
virtual InitIndicators	Initializes indicators and timeseries
virtual ValidationSettings	Checks the settings
Check Trailing Methods	
virtual CheckTrailingStopLong	Check Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#),
[RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

Period

Sets period of a moving average.

```
void Period(  
    int period    // period  
)
```

Parameters

period

[in] Period of a moving average.

Shift

Sets shift of a moving average.

```
void Shift(  
    int shift    // parameter  
)
```

Parameters

shift

[in] Shift of a moving average.

Method

Sets smoothing method of a moving average.

```
void Method(  
    ENUM_MA_METHOD method      // parameter  
)
```

Parameters

method

[in] Smoothing method (from [ENUM_MA_METHOD](#) enumeration) of a moving average.

Applied

Sets applied price of a moving average.

```
void Applied(  
    ENUM_APPLIED_PRICE applied // parameter  
)
```

Parameters

applied

[in] Applied price (from [ENUM_APPLIED_PRICE](#) enumeration) of a moving average.

InitIndicators

Initializes indicators and timeseries.

```
virtual bool InitIndicators(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to indicators and timeseries collection ([CExpert](#) class member).

Return Value

true - successful, otherwise - false.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function checks the period of moving average, the correct values are positive.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool CheckTrailingStopLong(  
    CPositionInfo* position,    // pointer  
    double& sl,                // reference  
    double& tp                 // reference  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the maximal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position. If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool CheckTrailingStopShort (
    CPositionInfo* position,    // pointer
    double& sl,                // reference
    double& tp                 // reference
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the minimal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position. If the calculated Stop Loss price is lower than base price and higher than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

CTrailingNone

CTrailingNone is a stub class. This class should be used at initialization of trailing object in CExpert class if your strategy does not use Trailing Stop.

Description

CTrailingNone class does not implement any Trailing Stop algorithm. The methods of checking Trailing Stop conditions always return false.

Declaration

```
class CTrailingNone: public CExpertTrailing
```

Title

```
#include <Expert\Trailing\TrailingNone.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertTrailing](#)

CTrailingNone

Class Methods by Groups

Check Trailing Methods	
virtual CheckTrailingStopLong	A stub method for checking Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	A stub method for checking Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertTrailing

[CheckTrailingStopLong](#), [CheckTrailingStopShort](#)

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool CheckTrailingStopLong(  
    CPositionInfo* position,    // pointer  
    double& sl,                // reference  
    double& tp                 // reference  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

The function always returns false.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool CheckTrailingStopShort (
    CPositionInfo* position,    // pointer
    double& sl,                // reference
    double& tp                 // reference
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

The function always returns false.

CTrailingPSAR

CTrailingPSAR is a class with implementation of Trailing Stop algorithm based on the values of Parabolic SAR indicator.

Description

CTrailingPSAR class implements the Trailing Stop algorithm based on the values of Parabolic SAR indicator of the previous (completed) bar.

Declaration

```
class CTrailingPSAR: public CExpertTrailing
```

Title

```
#include <Expert\Trailing\TrailingParabolicSAR.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertTrailing](#)

CTrailingPSAR

Class Methods by Groups

Initialization	
Step	Sets the value of step of Parabolic SAR indicator
Maximum	Sets the value of maximum of Parabolic SAR indicator
virtual InitIndicators	Initializes indicators and timeseries
Check Trailing Methods	
virtual CheckTrailingStopLong	Check conditions of trailing stop of a long position
virtual CheckTrailingStopShort	Check conditions of trailing stop of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#)

Step

Sets the value of step of Parabolic SAR indicator.

```
void Step(  
    double step    // parameter  
)
```

Parameters

step

[in] The value of step of Parabolic SAR indicator.

Maximum

Sets the value of maximum of Parabolic SAR indicator.

```
void Maximum(  
    double maximum    // parameter  
)
```

Parameters

maximum

[in] The value of maximum of Parabolic SAR indicator.

InitIndicators

Initializes indicators and timeseries.

```
virtual bool InitIndicators(  
    CIndicators* indicators // pointer  
)
```

Parameters

indicators

[in] Pointer to indicators and timeseries collection ([CExpert](#) class member).

Return Value

true - successful, otherwise - false.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool CheckTrailingStopLong(  
    CPositionInfo* position,    // pointer  
    double& sl,                // reference  
    double& tp                 // reference  
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the maximal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool CheckTrailingStopShort (
    CPositionInfo* position,    // pointer
    double& sl,                // reference
    double& tp                 // reference
)
```

Parameters

position

[in] Pointer to [CPositionInfo](#) object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the minimal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the calculated Stop Loss price is lower than base price and higher than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

Money Management classes

This section contains technical details of working with money and risk management classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) trading strategies.

MQL5 Standard Library (in terms of money and risk management classes) is placed in the terminal directory, in the Include\Expert\Money\ folder.

Class	Description
<u>CMoneyFixedLot</u>	This class implements money management algorithm based on trading with predefined fixed lot size.
<u>CMoneyFixedMargin</u>	This class implements money management algorithm based on trading with predefined fixed margin.
<u>CMoneyFixedRisk</u>	This class implements money management algorithm based on trading with predefined risk.
<u>CMoneyNone</u>	This class implements money management algorithm based on trading with minimal allowed lot size.
<u>CMoneySizeOptimized</u>	This class implements money management algorithm based on trading with variable lot size, depending on results of the previous deals.

CMoneyFixedLot

CMoneyFixedLot is the class designed to implement money management algorithm based on trading with predefined fixed lot size.

Description

CMoneyFixedLot implements money management algorithm based on trading with predefined fixed lot size.

Declaration

```
class CMoneyFixedLot: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedLot.mqh>
```

Inheritance hierarchy



Class Methods by Groups

Initialization	
Lots	Sets trading volume
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)

Lots

Sets trading volume (in lots).

```
void Lots(  
    double lots    // number of lots  
)
```

Parameters

lots

[in] Trading volume.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

Checks if the predefined fixed lot falls within the range of allowed values and is multiple of change step.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss order price.

Return Value

Trade volume for a long position.

Note

The function always returns the predefined fixed trade volume.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss order price.

Return Value

Trade volume for a short position.

Note

The function always returns the predefined fixed trade volume.

CMoneyFixedMargin

CMoneyFixedMargin is the class designed to implement money management algorithm based on trading with predefined fixed margin.

Description

CMoneyFixedMargin implements money management algorithm based on trading with predefined fixed margin.

Declaration

```
class CMoneyFixedMargin: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedMargin.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertMoney](#)

CMoneyFixedMargin

Class Methods by Groups

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertMoney

[Percent](#), [ValidationSettings](#), [CheckReverse](#), [CheckClose](#)

CheckOpenLong

Gets trade volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a long position with a fixed predefined margin. The margin is defined by Percent parameter of [CExpertMoney](#) base class.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function returns trade volume for a short position with a predefined fixed margin. The margin is defined by Percent parameter of [CExpertMoney](#) base class.

CMoneyFixedRisk

CMoneyFixedRisk is a class with implementation of money management algorithm with fixed predefined risk.

Description

CMoneyFixedRisk class implements the money management algorithm with fixed predefined risk.

Declaration

```
class CMoneyFixedRisk: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedRisk.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertMoney](#)

CMoneyFixedRisk

Class Methods by Groups

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertMoney

[Percent](#), [ValidationSettings](#), [CheckReverse](#)

CheckOpenLong

Gets trade volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a long position with a fixed predefined risk level. The risk is defined by Percent parameter of [CExpertMoney](#) base class.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function returns trade volume for a short position with a fixed predefined risk level. The risk is defined by Percent parameter of [CExpertMoney](#) base class.

CMoneyNone

CMoneyNone is a class with implementation of the "absence" of money and risk management.

Description

CMoneyNone class implements trading with minimal allowed lot.

Declaration

```
class CMoneyNone: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyNone.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertMoney](#)

CMoneyNone

Class Methods by Groups

Initialization	
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function always returns true.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function always returns the minimal lot size.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function always returns the minimal lot size.

CMoneySizeOptimized

CMoneySizeOptimized is a class with implementation of money and risk management algorithm depending on results of the previous deals.

Description

CMoneySizeOptimized implements the market entry algorithm with the lot size depending on results of the previous deals.

Declaration

```
class CMoneySizeOptimized: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneySizeOptimized.mqh>
```

Inheritance hierarchy

[CObject](#)

[CExpertBase](#)

[CExpertMoney](#)

CMoneySizeOptimized

Class Methods by Groups

Initialization	
DecreaseFactor	Sets the parameter value
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

Methods inherited from class CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)

DecreaseFactor

Sets the value of decrease factor.

```
void DecreaseFactor(  
    double decrease_factor    // decrease factor  
)
```

Parameters

decrease_factor

[in] Decrease factor.

Note

The DecreaseFactor defines the open position volume decreasing coefficient (compared with the volume of a previous position) for the case of consecutive loss trades.

ValidationSettings

Checks the settings.

```
virtual bool ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

If the value of decrease factor is negative, it returns false, otherwise it returns true.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double CheckOpenLong(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for long position.

Note

The function returns trade volume for a long position considering results of previous deals.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double CheckOpenShort(  
    double price,      // price  
    double sl          // Stop Loss price  
)
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a short position considering results of previous deals.

Classes for Creating Control Panels and Dialogs

This section contains technical details of working with classes for creation of controls panels and dialogs, as well as description of the relevant components of the MQL5 standard library.

The use of these classes will save time when developing custom interactive MQL5 applications, including Expert Advisors and indicators.

MQL5 Standard Library (in terms of classes for creation of control panels and dialogs) is placed in the terminal data folder, in the MQL5\Include\Controls.

The sample Expert Advisor, which illustrates the operation of these classes, can be found in MQL5\Expert\Examples\Controls.

Auxiliary structures	Description
<u>CRect</u>	Structure of the rectangular area
<u>CDateTime</u>	Structure for working with date and time

Base classes	Description
<u>CWnd</u>	Base class for all controls
<u>CWndObj</u>	Base class for controls and dialogs
<u>CWndContainer</u>	Base class for complex controls

Simple controls	Description
<u>CLabel</u>	Control, based on "Text label" graphic object
<u>CBmpButton</u>	Control, based on "Bitmap label" graphic object
<u>CButton</u>	Control, based on "Button" graphic object
<u>CEdit</u>	Control, based on "Edit field" graphic object
<u>CPanel</u>	Control, based on "Rectangle label"
<u>CPicture</u>	Control, based on "Bitmap label"

Complex controls	Description
<u>CScroll</u>	Base class of the scroll bar
<u>CScrollV</u>	Vertical scroll bar
<u>CScrollH</u>	Horizontal scroll bar
<u>CWndClient</u>	Base class of the client area with scroll bars
<u>CListView</u>	List view
<u>CComboBox</u>	Combo box

<u>CCheckBox</u>	Check box
<u>CCheckGroup</u>	Check group
<u>CRadioButton</u>	Radio button
<u>CRadioGroup</u>	Radio group
<u>CSpinEdit</u>	Increment/decrement field
<u>CDialog</u>	Dialog
<u>CAppDialog</u>	Main dialog of MQL5 application

CRect

CRect is a class of the rectangular area of the chart.

Description

CRect is a class of the area, it is defined by both coordinates of the upper-left and lower-right corners of a rectangle in Cartesian coordinates.

Declaration

```
class CRect
```

Title

```
#include <Controls\Rect.mqh>
```

Class Methods by Groups

Properties	
<u>Left</u>	Gets/sets the X coordinate of the upper-left corner
<u>Top</u>	Gets/sets the Y coordinate of the upper-left corner
<u>Right</u>	Gets/sets the X coordinate of the lower-right corner
<u>Bottom</u>	Gets/sets the Y coordinate of the lower-right corner
<u>Width</u>	Gets/sets the width
<u>Height</u>	Gets/sets the height
<u>SetBound</u>	Sets new coordinates of the area
<u>Move</u>	Performs the absolute displacement of the area coordinates
<u>Shift</u>	Performs the relative displacement (shift) of the area coordinates
<u>Contains</u>	Checks if the point is inside the area
Additional methods	
<u>Format</u>	Gets the area coordinates as a string

Left (Get Method)

Gets the X coordinate of the upper-left corner.

```
int Left()
```

Return Value

X coordinate of the upper-left corner.

Left (Set Method)

Sets the X coordinate of the upper-left corner.

```
void Left(  
    const int x    // x coordinate  
)
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Return Value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner.

```
int Top()
```

Return Value

Y coordinate of the upper-left corner.

Top (Set Method)

Sets the Y coordinate of the upper-left corner.

```
void Top(  
    const int y    // y coordinate  
)
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Return Value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner.

```
int Right()
```

Return Value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the X coordinate of the lower-right corner.

```
void Right(  
    const int x    // x coordinate  
)
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Return Value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner.

```
int Bottom()
```

Return Value

Y coordinate of the lower-right corner.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner.

```
void Bottom(  
    const int y    // y coordinate  
)
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Return Value

None.

Width (Get Method)

Gets the width of the area.

```
int Width()
```

Return Value

Width of the area.

Width (Set Method)

Sets new width of the area.

```
virtual bool Width(  
    const int w    // width  
)
```

Parameters

w

[in] New width.

Return Value

true - successful, otherwise - false.

Height (Get Method)

Gets the height of the area.

```
int Height()
```

Return Value

Height of the area.

Height (Set Method)

Sets new height of the area.

```
virtual bool Height(  
    const int h    // height  
)
```

Parameters

h

[in] New height.

Return Value

true - successful, otherwise - false.

SetBound

Sets new coordinates of the area using CRect class coordinates.

```
void SetBound(  
    const & CRect rect    // CRect class  
)
```

Return Value

None.

SetBound

Sets new coordinates of the area.

```
void SetBound(  
    const int l    // left  
    const int t    // top  
    const int r    // right  
    const int b    // bottom  
)
```

Parameters

l

[in] X coordinate of the upper-left corner.

t

[in] Y coordinate of the upper-left corner.

r

[in] X coordinate of the lower-right corner.

b

[in] Y coordinate of the lower-right corner.

Return Value

None.

Move

Performs the absolute displacement of the area coordinates.

```
void Move(  
    const int x,      // X coordinate  
    const int y      // Y coordinate  
)
```

Parameters

x

[in] New X coordinate.

y

[in] New Y coordinate.

Return Value

None.

Shift

Performs the relative displacement (shift) of the area coordinates.

```
void Shift(  
    const int dx,      // delta X  
    const int dy      // delta Y  
)
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

None.

Contains

Checks if the point is inside the CRect class area.

```
bool Contains(  
    const int x,      // X coordinate  
    const int y      // Y coordinate  
)
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Return Value

true - the point is inside the area (including borders), otherwise - false.

Format

Gets the area coordinates as a string.

```
string Format(  
    string & fmt,      // format  
    ) const
```

Parameters

fmt

[in] String with format.

Return Value

String with the area coordinates.

CDateTime

CDateTime is a structure for working with date and time.

Description

CDateTime is a structure derived from [MqlDateTime](#), it used for operation with date and time in controls.

Declaration

```
struct CDateTime
```

Title

```
#include <Tools\DateTime.mqh>
```

Methods

Properties	
MonthName	Gets month name
ShortMonthName	Gets short name of the month
DayName	Gets full name of the day in a week
ShortDayName	Gets short name of the day in a week
DaysInMonth	Gets number of days in month
Get/Set methods	
DateTime	Gets/sets date and time
Date	Sets date
Time	Sets time
Sec	Sets seconds
Min	Sets minutes
Hour	Sets hour
Day	Sets day of the month
Mon	Sets month
Year	Sets year
Additional methods	
SecDec	Subtracts specified number of seconds
SecInc	Adds specified number of seconds
MinDec	Subtracts specified number of minutes

<u>MinInc</u>	Adds specified number of minutes
<u>HourDec</u>	Subtracts specified number of hours
<u>HourInc</u>	Adds specified number of hours
<u>DayDec</u>	Subtracts specified number of days
<u>DayInc</u>	Adds specified number of days
<u>MonDec</u>	Subtracts specified number of months
<u>MonInc</u>	Adds specified number of months
<u>YearDec</u>	Subtracts specified number of years
<u>YearInc</u>	Adds specified number of years

MonthName

Gets month name.

```
string MonthName() const
```

Gets month name by index.

```
string MonthName(  
    const int    num        // month index  
) const
```

Parameters

num

[in] Month index (1-12).

Return Value

Full month name.

ShortMonthName

Gets short month name.

```
string ShortMonthName() const
```

Gets short month name by index.

```
string ShortMonthName(  
    const int    num        // month index  
) const
```

Parameters

num

[in] Month index (1-12).

Return Value

Short month name.

DayName

Gets full name of the day in a week.

```
string DayName() const
```

Gets full name of the day in a week by index.

```
string DayName (
    const int    num          // day index
) const
```

Parameters

num

[in] Day index (0-6).

Return Value

Full name of the day.

ShortDayName

Gets short name of the day in a week.

```
string ShortDayName() const
```

Gets short name of the day in a week by index.

```
string ShortDayName(  
    const int    num        // day index  
) const
```

Parameters

num

[in] Day index (0-6).

Return Value

Short name of the day.

DaysInMonth

Gets number of days in month.

```
int DaysInMonth() const
```

Return Value

Number of days in month.

DateTime (Get Method)

Gets date and time.

```
datetime DateTime()
```

Return Value

Value of [datetime](#) type.

DateTime (Set Method datetime)

Sets date and time with [datetime](#) type.

```
void DateTime(  
    const datetime    value    // date and time  
)
```

Parameters

value

[in] Value of [datetime](#) type.

Return Value

None.

DateTime (Set Method MqlDateTime)

Sets date and time with [MqlDateTime](#) type.

```
void DateTime(  
    const MqlDateTime &value    // date and time  
)
```

Parameters

value

[in] Value of [MqlDateTime](#) type.

Return Value

None.

Date (Set Method datetime)

Sets date with [datetime](#) type.

```
void Date(  
    const datetime    value    // date  
)
```

Parameters

value

[in] Value of [datetime](#) type.

Return Value

None.

Date (Set Method MqlDateTime)

Sets date with [MqlDateTime](#) type.

```
void Date(  
    const MqlDateTime &value    // date  
)
```

Parameters

value

[in] Value of [MqlDateTime](#) type.

Return Value

None.

Time (Set Method datetime)

Sets time with [datetime](#) type.

```
void Time(  
    const datetime    value    // time  
)
```

Parameters

value

[in] Value of [datetime](#) type.

Return Value

None.

Time (Set Method MqlDateTime)

Sets time with [MqlDateTime](#) type.

```
void Time(  
    const MqlDateTime &value    // time  
)
```

Parameters

value

[in] Value of [MqlDateTime](#) type.

Return Value

None.

Sec

Sets seconds.

```
void Sec(  
    const int value // seconds  
)
```

Parameters

value

[in] Seconds.

Return Value

None.

Min

Sets minutes.

```
void Min(  
    const int  value      // minutes  
)
```

Parameters

value

[in] Minutes.

Return Value

None.

Hour

Sets hour.

```
void Hour(  
    const int  value      // hour  
)
```

Parameters

value

[in] Hour.

Return Value

None.

Day

Sets day of the month.

```
void Day(  
    const int value // day of month  
)
```

Parameters

value

[in] Day of month.

Return Value

None.

Mon

Sets month.

```
void Day(  
    const int  value      // month  
)
```

Parameters

value

[in] Month.

Return Value

None.

Year

Sets year.

```
void Day(  
    const int  value      // year  
)
```

Parameters

value

[in] Year.

Return Value

None.

SecDec

Subtracts specified number of seconds.

```
void SecDec(  
    int    delta=1      // seconds  
)
```

Parameters

delta

[in] Seconds to subtract.

Return Value

None.

SecInc

Adds specified number of seconds.

```
void SecInc(  
    int    delta=1    // seconds  
)
```

Parameters

delta

[in] Seconds to add.

Return Value

None.

MinDec

Subtracts specified number of minutes.

```
void MinDec(  
    int    delta=1        // minutes  
)
```

Parameters

delta

[in] Minutes to subtract.

Return Value

None.

MinInc

Adds specified number of minutes.

```
void MinInc(  
    int    delta=1           // minutes  
)
```

Parameters

delta

[in] Minutes to add.

Return Value

None.

HourDec

Subtracts specified number of hours.

```
void HourDec (  
    int    delta=1        // hours  
)
```

Parameters

delta

[in] Hours to subtract.

Return Value

None.

HourInc

Adds specified number of hours.

```
void HourInc (  
    int    delta=1        // hours  
)
```

Parameters

delta

[in] Hours to add.

Return Value

None.

DayDec

Subtracts specified number of days.

```
void DayDec(  
    int    delta=1      // days  
)
```

Parameters

delta

[in] Days to subtract.

Return Value

None.

DayInc

Adds specified number of days.

```
void DayInc(  
    int    delta=1      // days  
)
```

Parameters

delta

[in] Days to add.

Return Value

None.

MonDec

Subtracts specified number of months.

```
void MonDec(  
    int    delta=1      // months  
)
```

Parameters

delta

[in] Months to subtract.

Return Value

None.

MonInc

Adds specified number of months.

```
void MonInc(  
    int    delta=1        // months  
)
```

Parameters

delta

[in] Months to add.

Return Value

None.

YearDec

Subtracts specified number of years.

```
void YearDec (
    int    delta=1      // years
)
```

Parameters

delta

[in] Years to subtract.

Return Value

None.

YearInc

Adds specified number of years.

```
void YearInc(  
    int    delta=1    // years  
)
```

Parameters

delta

[in] Years to add.

Return Value

None.

CWnd

CWnd is a base class for all Standard Library controls.

Description

CWnd class is the implementation of the base control class.

Declaration

```
class CWnd : public CObject
```

Title

```
#include <Controls\Wnd.mqh>
```

Inheritance hierarchy

CObject

CWnd

Direct descendants

CDragWnd, CWndContainer, CWndObj

Class Methods by Groups

Create and destroy	
<u>Create</u>	Creates control
<u>Destroy</u>	Destroys control
Chart event handlers	
<u>OnEvent</u>	Event handler of all chart events
<u>OnMouseEvent</u>	Event handler for the <u>CHARTEVENT_MOUSE_MOVE</u> event
Name	
<u>Name</u>	Gets control name
Access to container	
<u>ControlsTotal</u>	Gets the number of controls in the container
<u>Control</u>	Gets the control by index
<u>ControlFind</u>	Gets the control by ID
Geometry	
<u>Rect</u>	Gets the control coordinates

Left	Gets/sets the X coordinate of the upper-left corner
Top	Gets/sets the Y coordinate of the upper-left corner
Right	Gets/sets the X coordinate of the lower-right corner
Bottom	Gets/sets the Y coordinate of the lower-right corner
Width	Gets/sets the control width
Height	Gets/sets the control height
Move	Control absolute displacement
Shift	Control relative displacement
Resize	Changes control size
Contains	Checks if the point/control is inside the control area
Align	
Alignment	Sets alignment properties of the control
Align	Performs control alignment in the specified chart area
Identification	
Id	Gets/sets the control ID
State	
IsEnabled	Checks the control ability to respond to user's actions
Enable	Enables the control ability to respond to user's actions
Disable	Disables the control ability to respond to user's actions
IsVisible	Checks the visibility flag
Visible	Sets the visibility flag
Show	Shows the control
Hide	Hides the control
IsActive	Checks the control activity
Activate	Activates the control
Deactivate	Deactivates the control

State flags	
StateFlags	Gets/sets the control state flags
StateFlagsSet	Sets the control state flags
StateFlagsReset	Resets the control state flags
Properties flags	
PropFlags	Gets/sets the control properties flags
PropFlagsSet	Sets the control properties flags
PropFlagsReset	Resets the control properties flags
Mouse operations	
MouseX	Gets/saves the mouse X coordinate
MouseY	Gets/saves the mouse Y coordinate
MouseFlags	Gets/saves the mouse buttons state
MouseFocusKill	Kills mouse focus
Internal event handlers	
OnCreate	"Create" event handler
OnDestroy	"Destroy" event handler
OnMove	"Move" event handler
OnResize	"Resize" event handler
OnEnable	"Enable" event handler
OnDisable	"Disable" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler
OnClick	"Click" event handler
OnChange	"Change" event handler
Mouse event handlers	
OnMouseDown	"MouseDown" event handler
OnMouseUp	"MouseUp" event handler
Drag event handlers	
OnDragStart	"DragStart" event handler

OnDragProcess	"DragProcess" event handler
OnDragEnd	"DragEnd" event handler
Drag object	
DragObjectCreate	Creates drag object
DragObjectDestroy	Destroys drag object

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Create

Creates a control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,         // x1 coordinate  
    const int     y1,         // y1 coordinate  
    const int     x2,         // x2 coordinate  
    const int     y2         // y2 coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow, at which the control is created.

x1

[in] X coordinate of the upper-left corner.

y1

[in] Y coordinate of the upper-left corner.

x2

[in] X coordinate of the lower-right corner.

y2

[in] Y coordinate of the lower-right corner.

Return Value

true - successful, otherwise - false.

Note

Base class method only saves the creation parameters and always returns true.

Destroy

Destroys a control.

```
virtual bool Destroy()
```

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // event parameter  
    const double&  dparam,       // event parameter  
    const string&  sparam        // event parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type, passed by reference.

dparam

[in] Event parameter of [double](#) type, passed by reference.

sparam

[in] Event parameter of [string](#) type, passed by reference.

Return Value

true - event has been processed, otherwise - false.

OnMouseEvent

Mouse event handler (the [CHARTEVENT_MOUSE_MOVE](#) chart event).

```
virtual bool OnMouseEvent (  
    const int x,           // x coordinate  
    const int y,           // y coordinate  
    const int flags        // flags  
)
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Return Value

true - event has been processed, otherwise - false.

Name

Gets control name.

```
string Name() const
```

Return Value

Control name.

ControlsTotal

Gets the number of controls in the container.

```
int ControlsTotal() const
```

Return Value

Number of controls in container.

Note

The base class method does not have the container, it provides the access to container for its heirs and always returns 0.

Control

Gets the control from the container by index.

```
CWnd* Control(  
    const int ind    // index  
    ) const
```

Parameters

ind

[in] Control index.

Return Value

A pointer to the control.

Note

The base class method does not have the container, it provides the access to container for its heirs and always returns NULL.

ControlFind

Gets the control from container by specified ID.

```
virtual CWnd* ControlFind(  
    const long id      // ID  
)
```

Parameters

id

[in] Identifier of the control to find.

Return Value

Pointer to the control from the container.

Note

The base class method does not have the container, it provides the access to container for its heirs. If the specified ID matches the container's one, it returns a pointer to itself (this).

Rect

Gets the pointer to the CRect class object.

```
const CRect* Rect () const
```

Return Value

Pointer to the CRect class object.

Left (Get Method)

Gets the X coordinate of the upper-left corner of the control.

```
int Left()
```

Return Value

X coordinate of the upper-left corner of the control.

Left (Set Method)

Sets the X coordinate of the upper-left corner of the control.

```
void Left(  
    const int x    // coordinate  
)
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Return Value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner of the control.

```
int Top()
```

Return Value

Y coordinate of the upper-left corner of the control.

Top (Set Method)

Sets the Y coordinate of the upper-left corner of the control.

```
void Top(  
    const int y    // y coordinate  
)
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Return Value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner of the control.

```
int Right()
```

Return Value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the X coordinate of the lower-right corner of the control.

```
void Right(  
    const int x    // x coordinate  
)
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Return Value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner of the control.

```
int Bottom()
```

Return Value

Y coordinate of the lower-right corner of the control.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner of the control.

```
void Bottom(  
    const int y    // y coordinate  
)
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Return Value

None.

Width (Get Method)

Gets the control width.

```
int Width()
```

Return Value

Width of the control.

Width (Set Method)

Sets the width of the control.

```
virtual bool Width(  
    const int w    // width  
)
```

Parameters

w

[in] New width.

Return Value

true - successful, otherwise - false.

Height (Get Method)

Gets the control height.

```
int Height()
```

Return Value

Height of the control.

Height (Set Method)

Sets new height of the control.

```
virtual bool Height(  
    const int h    // height  
)
```

Parameters

h

[in] New height.

Return Value

true - successful, otherwise - false.

Move

Performs absolute displacement of a control.

```
void Move(  
    const int x,      // x coordinate  
    const int y      // y coordinate  
)
```

Parameters

x

[in] New X coordinate of the upper-left point.

y

[in] New Y coordinate of the upper-left point.

Return Value

true - successful, otherwise - false.

Shift

Performs the relative shift of a control.

```
void Shift(  
    const int dx,      // delta X  
    const int dy      // delta Y  
)
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

true - successful, otherwise - false.

Resize

Sets a size of the control.

```
virtual bool  Resize(  
    const int  w,  
    const int  h  
)
```

Parameters

w

[in] New width.

h

[in] New height.

Return Value

true - successful, otherwise - false.

Contains

Checks if the point is inside the control area of the chart.

```
bool Contains(  
    const int x,      // X coordinate  
    const int y      // Y coordinate  
)
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Return Value

true - the point is inside the area (including borders), otherwise - false.

Contains

Checks if the specified control is inside the control area of the chart.

```
bool Contains(  
    const CWnd* control // pointer  
) const
```

Parameters

control

[in] Object pointer.

Return Value

true - the specified control is inside the area (including borders), otherwise - false.

Alignment

Sets alignment parameters of the control.

```
void Alignment(  
    const int  flags,      // flags  
    const int  left,       // offset  
    const int  top,        // offset  
    const int  right,      // offset  
    const int  bottom     // offset  
)
```

Parameters

flags

[in] Alignment flags.

left

[in] Fixed offset from the left border.

top

[in] Fixed offset from the top border.

right

[in] Fixed offset from the right border.

bottom

[in] Fixed offset from the bottom border.

Return Value

None.

Note

Alignement flags:

```
enum WND_ALIGN_FLAGS  
{  
    WND_ALIGN_NONE=0,           // no align  
    WND_ALIGN_LEFT=1,          // align left  
    WND_ALIGN_TOP=2,           // align top  
    WND_ALIGN_RIGHT=4,         // align right  
    WND_ALIGN_BOTTOM=8,        // align bottom  
    WND_ALIGN_WIDTH = WND_ALIGN_LEFT|WND_ALIGN_RIGHT, // align width  
    WND_ALIGN_HEIGHT=WND_ALIGN_TOP|WND_ALIGN_BOTTOM,  // align height  
    WND_ALIGN_CLIENT=WND_ALIGN_WIDTH|WND_ALIGN_HEIGHT, // align height and width  
}
```


Align

Performs control alignment in the specified chart area.

```
virtual bool Align(  
    const CRect* rect    // pointer  
)
```

Parameters

rect

[in] Pointer to the object with chart area coordinates.

Return Value

true - successful, otherwise - false.

Note

The alignment parameters must be specified (no alignment by default).

Id (Get Method)

Gets the control ID.

```
long Id() const
```

Return Value

The control identifier.

Id (Set Method)

Sets the value of the control ID.

```
virtual long Id(  
    const long id    // identifier  
)
```

Parameters

id

[in] New value of the control identifier.

Return Value

None.

IsEnabled

Checks the control ability to respond to user's actions.

```
bool IsEnabled() const
```

Return Value

true - control is enabled, otherwise - false.

Enable

Enables the control ability to respond to user's actions.

```
virtual bool Enable()
```

Return Value

true - successful, otherwise - false.

Note

If the control is enabled, it is able to process the external events.

Disable

Disables the control ability to respond to user's actions.

```
virtual bool Disable()
```

Return Value

true - successful, otherwise - false.

Note

The disabled control is not able to process the external events.

IsVisible

Checks whether the control is visible.

```
bool IsVisible() const
```

Return Value

true - control is shown on the chart, otherwise - false.

Visible

Sets the visibility flag.

```
virtual bool Visible(  
    const bool flag    // flag  
)
```

Parameters

flag

[in] New flag.

Return Value

true - successful, otherwise - false.

Show

Shows the control.

```
virtual bool Show()
```

Return Value

true - successful, otherwise - false.

Hide

Hides the control.

```
virtual bool Hide()
```

Return Value

true - successful, otherwise - false.

IsActive

Gets a value indicating whether the control is active.

```
bool IsActive() const
```

Return Value

true - control is active, otherwise - false.

Activate

Activates the control.

```
virtual bool Activate()
```

Return Value

true - successful, otherwise - false.

Note

The control becomes active when the mouse cursor is hovering over it.

Deactivate

Deactivates the control.

```
virtual bool Deactivate()
```

Return Value

true - successful, otherwise - false.

Note

The control becomes inactive when the mouse cursor is out off the control.

StateFlags (Get Method)

Gets the control state flags.

```
int StateFlags()
```

Return Value

The control state flags.

StateFlags (Set Method)

Sets the control state flags.

```
virtual void StateFlags (  
    const int flags    // flags  
)
```

Parameters

flags

[in] New control state flags.

Return Value

None.

StateFlagsSet

Sets the control state flags.

```
virtual void StateFlagsSet(  
    const int flags    // flags  
)
```

Parameters

flags

[in] Flags to set.

Return Value

None.

StateFlagsReset

Resets the control state flags.

```
virtual void StateFlagsReset(  
    const int flags    // flags  
)
```

Parameters

flags

[in] Flags to reset.

Return Value

None.

PropFlags (Get Method)

Gets the control properties flags.

```
void PropFlags(  
    const int flags    // flags  
)
```

Return Value

The control properties flags.

PropFlags (Set Method)

Sets the control properties flags.

```
virtual void PropFlags(  
    const int flags    // flags  
)
```

Parameters

flags

[in] New flags.

Return Value

None.

PropFlagsSet

Sets the control properties flags.

```
virtual void PropFlagsSet(  
    const int flags    // flags  
)
```

Parameters

flags

[in] Flags to set.

Return Value

None.

PropFlagsReset

Resets the control properties flags.

```
virtual void PropFlagsReset(  
    const int flags    // flags  
)
```

Parameters

flags

[in] Flags to reset.

Return Value

None.

MouseX (Set Method)

Saves the mouse X coordinate.

```
void MouseX(  
    const int value    // coordinate  
)
```

Parameters

value

[in] The X coordinate of the mouse.

Return Value

None.

MouseX (Get Method)

Gets the saved X coordinate of the mouse.

```
int MouseX()
```

Return Value

The X coordinate of the mouse.

MouseY (Set Method)

Saves the mouse Y coordinate.

```
void MouseY(  
    const int value    // coordinate  
)
```

Parameters

value

[in] The Y coordinate of the mouse.

Return Value

None.

MouseY (Get Method)

Gets the saved Y coordinate of the mouse.

```
int MouseY()
```

Return Value

The Y coordinate of the mouse.

MouseFlags (Set Method)

Saves the state of mouse buttons.

```
virtual void MouseFlags(  
    const int value    // state  
)
```

Parameters

value

[in] State of mouse buttons.

Return Value

None.

MouseFlags (Get Method)

Gets the saved state of mouse buttons.

```
int MouseFlags()
```

Return Value

State of mouse buttons.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates the control.

```
bool MouseFocusKill(  
    const long id=CONTROLS_INVALID_ID    // id  
)
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Return Value

The control deactivation result.

OnCreate

The virtual handler of the internal "Create" event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDestroy

The virtual handler of the internal "Destroy" event.

```
virtual bool OnDestroy()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnMove

The virtual handler of the internal "Move" event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnResize

The virtual handler of the internal "Resize" event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnEnable

The virtual handler of the internal "Enable" event (if enabled, it can respond to user interaction).

```
virtual bool OnEnable()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDisable

The virtual handler of the internal "Disable" event (if disabled, it cannot respond to user interaction).

```
virtual bool OnDisable()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnShow

The virtual handler of the internal "Show" event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHide

The virtual handler of the internal "Hide" event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnActivate

The virtual handler of the internal "Activate" event.

```
virtual bool OnActivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The virtual handler of the internal "Deactivate" event.

```
virtual bool OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnClick

The virtual handler of the internal "Click" event (mouse click).

```
virtual bool OnClick()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of the internal "Change" event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnMouseDown

The virtual handler of the "MouseDown" (mouse click) event.

```
virtual bool OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The virtual handler of the "MouseUp" (mouse button release) event.

```
virtual bool OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

OnDragStart

The virtual handler of the "DragStart" (start dragging) event.

```
virtual bool OnDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "DragStart" event occurs at the start of control dragging operation.

OnDragProcess

The virtual handler of the "DragProcess" (dragging) event.

```
virtual bool OnDragProcess(  
    const int x,      // x coordinate  
    const int y      // y coordinate  
)
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "DragProcess" event occurs when the control is dragged.

OnDragEnd

The virtual handler of the "DragEnd" event.

```
virtual bool OnDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "DragEnd" event occurs when control drag process is finished.

DragObjectCreate

Creates drag object.

```
virtual bool DragObjectCreate ()
```

Return Value

true - successful, otherwise - false.

Note

true - event processed, otherwise - false.

DragObjectDestroy

Destroys drag object.

```
virtual bool DragObjectDestroy()
```

Return Value

true - successful, otherwise - false.

CWndObj

CWndObj is a base class for simple controls (based on chart objects) of the Standard Library.

Description

CWndObj class implements base methods of the simple control.

Declaration

```
class CWndObj : public CWnd
```

Title

```
#include <Controls\WndObj.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

CWndObj

Direct descendants

[CBmpButton](#), [CButton](#), [CEdit](#), [CLabel](#), [CPanel](#), [CPicture](#)

Class Methods by Groups

Chart events processing	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/sets the OBJPROP_TEXT property of the chart object
Color	Gets/sets the OBJPROP_COLOR property of the chart object
ColorBackground	Gets/sets the OBJPROP_BGCOLOR property of the chart object
ColorBorder	Gets/sets the OBJPROP_BORDER_COLOR property of the chart object
Font	Gets/sets the OBJPROP_FONT property of the chart object
FontSize	Gets/sets the OBJPROP_FONTSIZE property of the chart object
ZOrder	Gets/sets the OBJPROP_ZORDER property of the chart object

Chart objects event handlers	
OnObjectCreate	CHARTEVENT_OBJECT_CREATE event handler
OnObjectChange	CHARTEVENT_OBJECT_CHANGE event handler
OnObjectDelete	CHARTEVENT_OBJECT_DELETE event handler
OnObjectDrag	CHARTEVENT_OBJECT_DRAG event handler
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnDestroy	"Destroy" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Create](#), [Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type, passed by reference.

dparam

[in] Event parameter of [double](#) type, passed by reference.

sparam

[in] Event parameter of [string](#) type, passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets the [OBJPROP_TEXT](#) (text) property of the chart object.

```
string Text()
```

Return Value

The value of the [OBJPROP_TEXT](#) property.

Text (Set method)

Sets the [OBJPROP_TEXT](#) (text) property of the chart object.

```
bool Text (  
    const string value    // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_TEXT](#) property.

Return Value

true - successful, otherwise - false.

Color (Get method)

Gets the [OBJPROP_COLOR](#) (color) property of the chart object.

```
color Color()
```

Return Value

The value of the [OBJPROP_COLOR](#) property.

Color (Set method)

Sets the [OBJPROP_COLOR](#) (color) property of the chart object.

```
bool Color(  
    const color value    // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_COLOR](#) property.

Return Value

true - successful, otherwise - false.

ColorBackground (Get method)

Gets the [OBJPROP_BGCOLOR](#) (background color) of the chart object.

```
color ColorBackground()
```

Return Value

The value of the [OBJPROP_BGCOLOR](#) property.

ColorBackground (Set method)

Sets the [OBJPROP_BGCOLOR](#) (background color) property of the chart object.

```
bool ColorBackground(  
    const color value    // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_BGCOLOR](#) property.

Return Value

true - successful, otherwise - false.

ColorBorder (Get method)

Gets the [OBJPROP_BORDER_COLOR](#) (border color) property of the chart object.

```
color ColorBorder ()
```

Return Value

The value of the [OBJPROP_BORDER_COLOR](#) property.

ColorBorder (Set method)

Sets the [OBJPROP_BORDER_COLOR](#) (border color) property of the chart object.

```
bool ColorBorder (
    const color value    // value
)
```

Parameters

value

[in] New value of the [OBJPROP_BORDER_COLOR](#) property.

Return Value

true - successful, otherwise - false.

Font (Get method)

Gets the [OBJPROP_FONT](#) (font) property of the chart object.

```
string Font()
```

Return Value

The value of the [OBJPROP_FONT](#) property.

Font (Set method)

Sets the [OBJPROP_FONT](#) (font) property of the chart object.

```
bool Font(  
    const string value    // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_FONT](#) property.

Return Value

true - successful, otherwise - false.

FontSize (Get method)

Gets the [OBJPROP_FONTSIZE](#) (font size) property of the chart object.

```
int FontSize()
```

Return Value

The value of the [OBJPROP_FONTSIZE](#) property.

FontSize (Set method)

Sets the [OBJPROP_FONTSIZE](#) (font size) property of the chart object.

```
bool FontSize(  
    const int value    // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_FONTSIZE](#) property.

Return Value

true - successful, otherwise - false.

ZOrder (Get method)

Gets the [OBJPROP_ZORDER](#) property of the chart object.

```
long ZOrder()
```

Return Value

The value of the [OBJPROP_ZORDER](#) property.

ZOrder (Set method)

Sets the [OBJPROP_ZORDER](#) property of the chart object.

```
bool ZOrder(  
    const long value // value  
)
```

Parameters

value

[in] New value of the [OBJPROP_ZORDER](#) property.

Return Value

true - successful, otherwise - false.

OnObjectCreate

The virtual handler of chart object [CHARTEVENT_OBJECT_CREATE](#) event.

```
virtual bool OnObjectCreate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnObjectChange

The virtual handler of chart object [CHARTEVENT_OBJECT_CHANGE](#) event.

```
virtual bool OnObjectChange ()
```

Return Value

true - event processed, otherwise - false.

OnObjectDelete

The virtual handler of chart object [CHARTEVENT_OBJECT_DELETE](#) event.

```
virtual bool OnObjectDelete()
```

Return Value

true - event processed, otherwise - false.

OnObjectDrag

The virtual handler of chart object [CHARTEVENT_OBJECT_DRAG](#) event.

```
virtual bool OnObjectDrag()
```

Return Value

true - event processed, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the [OBJPROP_TEXT](#) property) event.

```
virtual bool OnSetText()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetColor

The virtual handler of control "SetColor" (change of the [OBJPROP_COLOR](#) property) event.

```
virtual bool OnSetColor()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the [OBJPROP_BGCOLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetFont

The virtual handler of "SetFont" (change of the [OBJPROP_FONT](#) property) event.

```
virtual bool OnSetFont()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetFontSize

The virtual handler of "SetFontSize" (change of the [OBJPROP_FONTSIZE](#) property) event.

```
virtual bool OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetZOrder

The virtual handler of "SetZOrder" (change of the [OBJPROP_ZORDER](#) property) event.

```
virtual bool OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDestroy

The virtual handler of control "Destroy" internal event.

```
virtual bool OnDestroy()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

CWndContainer

CWndContainer is a base class for a complex control (containing dependent controls) of the Standard library.

Description

CWndContainer class implements base methods of the complex control.

Declaration

```
class CWndContainer : public CWnd
```

Title

```
#include <Controls\WndContainer.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

CWndContainer

Direct descendants

[CCheckBox](#), [CComboBox](#), [CDateDropList](#), [CDatePicker](#), [CDialog](#), [CRadioButton](#), [CScroll](#), [CSpinEdit](#), [CWndClient](#)

Class Methods by Groups

Destroy	
Destroy	Destroys all the container controls
Chart event handlers	
OnEvent	Event handler of all chart events
OnMouseEvent	The CHARTEVENT_MOUSE_MOVE event handler
Access to container	
ControlsTotal	Gets the number of controls in the container
Control	Gets control by index
ControlFind	Gets control by ID
Add/Delete	
Add	Adds control to a group (container)
Delete	Deletes control from a group (container)
Geometry	

Move	Performs a absolute displacement of an element group
Shift	Performs a relative displacement of an element group
Identification	
Id	Sets the ID for all controls of the container
State	
Enable	Enables all controls of the container
Disable	Disables all controls of the container
Show	Shows all controls of the container
Hide	Hides all controls of the container
Mouse operations	
MouseFocusKill	Kills mouse focus
File operations	
Save	Saves container information to file
Load	Loads container information from file
Internal event handlers	
OnResize	"Resize" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Create](#), [Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Destroy

Destroys all the container controls.

```
virtual bool Destroy()
```

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,      // parameter  
    const double&  dparam,      // parameter  
    const string&  sparam       // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type, passed by reference.

dparam

[in] Event parameter of [double](#) type, passed by reference.

sparam

[in] Event parameter of [string](#) type, passed by reference.

Return Value

true - event processed, otherwise - false.

OnMouseEvent

Mouse event handler.

```
virtual bool OnMouseEvent (  
    const int x,           // x coordinate  
    const int y,           // y coordinate  
    const int flags        // flags  
)
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Return Value

true - event processed, otherwise - false.

ControlsTotal

Gets the number of controls in the container.

```
int ControlsTotal() const
```

Return Value

Number of controls in the container.

Control

Gets control from the container by index.

```
CWnd* Control(  
    const int ind    // index  
) const
```

Parameters

ind

[in] Index of the control needed.

Return Value

Pointer to the control, otherwise NULL if the control is not found.

ControlFind

Gets control from the container by identifier.

```
virtual CWnd* ControlFind(  
    const long id    // id  
)
```

Parameters

id

[in] Control ID.

Return Value

Pointer to the control, otherwise NULL if the control is not found.

Add

Adds a control to a group (container).

```
bool Add(  
    CWnd& control    // reference  
)
```

Parameters

control

[in] Control to add, passed by reference.

Return Value

true - successful, otherwise - false.

Delete

Deletes control from a group (container).

```
bool Delete(  
    CWnd& control    // reference  
)
```

Parameters

control

[in] Control to delete, passed by reference.

Return Value

true - successful, otherwise - false.

Move

Performs an absolute displacement of all controls of the container.

```
virtual bool Move(  
    const int x,      // X coordinate  
    const int y      // Y coordinate  
)
```

Parameters

x

[in] New X coordinate of the upper-left corner.

y

[in] New Y coordinate of the upper-left corner.

Return Value

true - successful, otherwise - false.

Shift

Performs the relative shift of the coordinates for all controls of the container.

```
virtual bool Shift(  
    const int dx,      // delta X  
    const int dy      // delta Y  
)
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

true - successful, otherwise - false.

Id

Sets the ID for all controls of the container.

```
virtual long Id(  
    const long id    // identifier  
)
```

Parameters

id

[in] Base group identifier.

Return Value

Number of identifiers used by container controls.

Enable

Enables all the controls of the container.

```
virtual bool Enable()
```

Return Value

true - successful, otherwise - false.

Disable

Disables all controls of the container.

```
virtual bool Disable()
```

Return Value

true - successful, otherwise - false.

Show

Shows all controls of the container.

```
virtual bool Show()
```

Return Value

true - successful, otherwise - false.

Hide

Hides all controls of the container.

```
virtual bool Hide()
```

Return Value

true - successful, otherwise - false.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates all controls in the container.

```
bool MouseFocusKill(  
    const long id=CONTROLS_INVALID_ID    // id  
)
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Return Value

The controls deactivation result.

Save

Saves container information to file.

```
virtual bool Save(  
    const int file_handle    // handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened for writing.

Return Value

true - successful, otherwise - false.

Load

Loads container information from file

```
virtual bool Load(  
    const int file_handle // handle  
)
```

Parameters

file_handle

[in] Handle of the binary file previously opened for reading.

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnActivate

The virtual handler of control "Activate" internal event.

```
virtual bool OnActivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The virtual handler of control "Deactivate" internal event.

```
virtual bool OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

CLabel

CLabel is class of the simple control based on "Text label" chart object.

Description

CLabel is intended for creation of simple non-editable text labels.

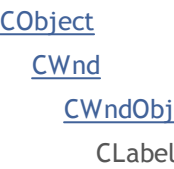
Declaration

```
class CLabel : public CWndObj
```

Title

```
#include <Controls\Label.mqh>
```

Inheritance hierarchy



Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with text label:

```
//+-----+
//|                                     ControlsLabel.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CLabel"
#include <Controls\Dialog.mqh>
#include <Controls\Label.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
```



```

#define INDENT_LEFT           (11)      // indent from left (with allowa
#define INDENT_TOP            (11)      // indent from top (with allowa
#define INDENT_RIGHT          (11)      // indent from right (with allowa
#define INDENT_BOTTOM         (11)      // indent from bottom (with allowa
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT         (20)     // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT           (20)     // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH           (150)     // size by X coordinate
#define LIST_HEIGHT           (179)    // size by Y coordinate
#define RADIO_HEIGHT          (56)     // size by Y coordinate
#define CHECK_HEIGHT          (93)     // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CLabel          m_label;           // CLabel object
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool     Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool     OnEvent(const int id,const long &lparam,const double &dparam,const
protected:
    //--- create dependent controls
    bool             CreateLabel(void);
    //--- handlers of the dependent controls events
    void             OnClickLabel(void);
};
//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)

EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

```



```

//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateLabel())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "CLabel" |
//+-----+
bool CControlsDialog::CreateLabel(void)
{
//--- coordinates
    int x1=INDENT_RIGHT;
    int y1=INDENT_TOP+CONTROLS_GAP_Y;
    int x2=x1+100;
    int y2=y1+20;
//--- create
    if(!m_label.Create(m_chart_id,m_name+"Label",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!m_label.Text("Label"))
        return(false);
    if(!Add(m_label))
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- create application dialog

```



```

    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates new CLabel control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // x1 coordinate  
    const int     y1,        // y1 coordinate  
    const int     x2,        // x2 coordinate  
    const int     y2         // y2 coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the [OBJPROP_TEXT](#) property) event.

```
virtual bool OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the [OBJPROP_COLOR](#) property) event.

```
virtual bool OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the [OBJPROP_FONT](#) property) event.

```
virtual bool OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the [OBJPROP_FONTSIZE](#) property) event.

```
virtual bool OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

CBmpButton

CBmpButton is class of the simple control based on "Bitmap label" chart object.

Description

CBmpButton is intended for creation of buttons with graphic image.

Declaration

```
class CBmpButton : public CWndObj
```

Title

```
#include <Controls\BmpButton.mqh>
```

Inheritance hierarchy

- CObject
 - CWnd
 - CWndObj
 - CBmpButton

Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

Properties	
Border	Gets/sets the "Border" property of the control
BmpNames	Sets the name of bmp files of the control
BmpOffName	Gets/sets the name of bmp file for the OFF state
BmpOnName	Gets/sets the name of bmp file for the ON state
BmpPassiveName	Gets/sets the name of bmp file for the passive state
BmpActiveName	Gets/sets the name of bmp file for the active state
State	
Pressed	Gets/sets the state of the control
Locking	Gets/sets the "Locking" property of the control
Internal event handlers	
OnSetZOrder	"SetZOrder" internal event handler
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnChange	"Change" internal event handler
OnActivate	"Activate" internal event handler
OnDeactivate	"Deactivate" internal event handler
OnMouseDown	"MouseDown" internal event handler
OnMouseUp	"MouseUp" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with Bitmap label:

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CBmpButton"
#include <Controls\Dialog.mqh>
#include <Controls\BmpButton.mqh>

//+-----+
//| defines |
//+-----+

//--- indents and gaps
#define INDENT_LEFT      (11)      // indent from left (with allowance)
#define INDENT_TOP       (11)      // indent from top (with allowance)
#define INDENT_RIGHT     (11)      // indent from right (with allowance)
#define INDENT_BOTTOM    (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X   (5)       // gap by X coordinate
#define CONTROLS_GAP_Y   (5)       // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT       (20)      // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH       (150)     // size by X coordinate
#define LIST_HEIGHT       (179)     // size by Y coordinate
#define RADIO_HEIGHT      (56)      // size by Y coordinate
#define CHECK_HEIGHT      (93)      // size by Y coordinate

//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CBmpButton      m_bmpbutton1;      // CBmpButton object
    CBmpButton      m_bmpbutton2;      // CBmpButton object

public:
    CControlsDialog(void);
```



```

~CControlsDialog(void);

//--- create
virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool          CreateBmpButton1(void);
    bool          CreateBmpButton2(void);
    //--- handlers of the dependent controls events
    void          OnClickBmpButton1(void);
    void          OnClickBmpButton2(void);
};

//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_bmpbutton1,OnClickBmpButton1)
ON_EVENT(ON_CLICK,m_bmpbutton2,OnClickBmpButton2)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateBmpButton1())
        return(false);
    if(!CreateBmpButton2())
        return(false);
//--- succeed
    return(true);
}
//+-----+

```



```

//| Create the "BmpButton1" button |
//+-----+
bool CControlsDialog::CreateBmpButton1(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_bmpbutton1.Create(m_chart_id,m_name+"BmpButton1",m_subwin,x1,y1,x2,y2))
        return(false);
//--- sets the name of bmp files of the control CBmpButton
    m_bmpbutton1.BmpNames("\\Images\\euro.bmp", "\\Images\\dollar.bmp");
    if(!Add(m_bmpbutton1))
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "BmpButton2" fixed button |
//+-----+
bool CControlsDialog::CreateBmpButton2(void)
{
//--- coordinates
    int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_bmpbutton2.Create(m_chart_id,m_name+"BmpButton2",m_subwin,x1,y1,x2,y2))
        return(false);
//--- sets the name of bmp files of the control CBmpButton
    m_bmpbutton2.BmpNames("\\Images\\euro.bmp", "\\Images\\dollar.bmp");
    if(!Add(m_bmpbutton2))
        return(false);
    m_bmpbutton2.Locking(true);
//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnClickBmpButton1(void)
{
    Comment(__FUNCTION__);
}
//+-----+
//| Event handler |

```



```

//+-----+
void CControlsDialog::OnClickBmpButton2(void)
{
    if(m_bmpbutton2.Pressed())
        Comment(__FUNCTION__+" State of the control is: On");
    else
        Comment(__FUNCTION__+" State of the control is: Off");
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //---
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates new CBmpButton control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // x1 coordinate  
    const int     y1,        // y1 coordinate  
    const int     x2,        // x2 coordinate  
    const int     y2         // y2 coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int Border() const
```

Return Value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool Border(  
    const int value    // value  
)
```

Parameters

value

[in] New value of the "Border" property.

Return Value

true - successful, otherwise - false.

BmpNames

Sets the name of bmp files of the control

```
bool BmpNames(  
    const string off="",      // file name  
    const string on=""        // file name  
)
```

Parameters

off=""

[in] Name of bmp file for OFF state.

on=""

[in] Name of bmp file for ON state.

Return Value

true - successful, otherwise - false.

BmpOffName (Get method)

Gets the name of bmp file for OFF state.

```
string BmpOffName() const
```

Return Value

Name of bmp file for OFF state.

BmpOffName (Set method)

Sets the name of bmp file for OFF state.

```
bool BmpOffName (  
    const string name    // file name  
)
```

Parameters

name

[in] Name of bmp file for OFF state.

Return Value

true - successful, otherwise - false.

BmpOnName (Get method)

Gets the name of bmp file for ON state.

```
string BmpOnName() const
```

Return Value

Name of bmp file for ON state.

BmpOnName (Set method)

Sets the name of bmp file for ON state.

```
bool BmpOnName (
    const string name    // file name
)
```

Parameters

name

[in] Name of bmp file for ON state.

Return Value

true - successful, otherwise - false.

BmpPassiveName (Get method)

Gets the name of bmp file for the control passive state.

```
string BmpPassiveName() const
```

Return Value

Name of bmp file for the control passive state.

BmpPassiveName (Set method)

Sets the name of bmp file for the passive state.

```
bool BmpPassiveName(  
    const string name    // file name  
)
```

Parameters

name

[in] Name of bmp file for the control passive state.

Return Value

true - successful, otherwise - false.

BmpActiveName (Get method)

Gets the name of bmp file for the active state.

```
string BmpActiveName() const
```

Return Value

Name of bmp file for the active state.

Note

The control becomes active when the mouse cursor is hovering over it.

BmpActiveName (Set method)

Sets the name of bmp file for the active state.

```
bool BmpActiveName(  
    const string name    // file name  
)
```

Parameters

name

[in] Name of bmp file for the active state.

Return Value

true - successful, otherwise - false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool Pressed() const
```

Return Value

Control state.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool Pressed(  
    const bool pressed // state  
)
```

Parameters

pressed

[in] New control state.

Return Value

true - successful, otherwise - false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool Locking() const
```

Return Value

The value of "Locking" property.

Locking (Set method)

Sets new value of the "Locking" property of the control.

```
void Locking(  
    const bool locking    // value  
)
```

Parameters

locking

[in] New value of "Locking" property.

Return Value

None.

OnSetZOrder

The virtual handler of control "SetZOrder" (change of the [OBJPROP_ZORDER](#) property) event.

```
virtual bool OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

OnActivate

The virtual handler of control "Activate" internal event.

```
virtual bool OnActivate()
```

Return Value

true - event processed, otherwise - false.

OnDeactivate

The virtual handler of control "Deactivate" internal event.

```
virtual bool OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

OnMouseDown

The virtual handler of control "MouseDown" event.

```
virtual bool OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

OnMouseUp

The virtual handler of control "MouseUp" (mouse button release) event.

```
virtual bool OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

CButton

CButton is a class of a simple control based on "Button" chart object.

Description

CButton class is intended for creation of simple buttons.

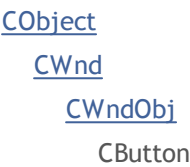
Declaration

```
class CButton : public CWndObj
```

Title

```
#include <Controls\Button.mqh>
```

Inheritance hierarchy



Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

State	
Pressed	Gets/sets the "Pressed" property
Locking	Gets/sets the "Locking" property
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnMouseDown	"MouseDown" internal event handler
OnOnMouseUp	"MouseUp" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with button:

```
//+-----+
//|                                     ControlsButton.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
```



```

//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CButton"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)           // indent from left (with allowa
#define INDENT_TOP            (11)           // indent from top (with allowa
#define INDENT_RIGHT          (11)           // indent from right (with allowa
#define INDENT_BOTTOM         (11)           // indent from bottom (with allo
#define CONTROLS_GAP_X        (5)            // gap by X coordinate
#define CONTROLS_GAP_Y        (5)            // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)          // size by X coordinate
#define BUTTON_HEIGHT         (20)           // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT           (20)           // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH           (150)          // size by X coordinate
#define LIST_HEIGHT           (179)          // size by Y coordinate
#define RADIO_HEIGHT          (56)           // size by Y coordinate
#define CHECK_HEIGHT          (93)           // size by Y coordinate
//+-----+
//| Class CControlsDialog                                     |
//| Usage: main dialog of the Controls application           |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CButton      m_button1;                  // the button object
    CButton      m_button2;                  // the button object
    CButton      m_button3;                  // the fixed button object

public:
                                CControlsDialog(void);
                                ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls

```



```

    bool        CreateButton1(void);
    bool        CreateButton2(void);
    bool        CreateButton3(void);
    ///--- handlers of the dependent controls events
    void        OnClickButton1(void);
    void        OnClickButton2(void);
    void        OnClickButton3(void);
};

//+-----+
//| Event Handling                                     |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_button1,OnClickButton1)
ON_EVENT(ON_CLICK,m_button2,OnClickButton2)
ON_EVENT(ON_CLICK,m_button3,OnClickButton3)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor                                     |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor                                     |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create                                     |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    ///--- create dependent controls
    if(!CreateButton1())
        return(false);
    if(!CreateButton2())
        return(false);
    if(!CreateButton3())
        return(false);
    ///--- succeed
    return(true);
}
//+-----+
//| Create the "Button1" button                                     |
//+-----+
bool CControlsDialog::CreateButton1(void)

```



```

{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_button1.Create(m_chart_id,m_name+"Button1",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!m_button1.Text("Button1"))
        return(false);
    if(!Add(m_button1))
        return(false);
//--- succeed
    return(true);
}

//+-----+
//| Create the "Button2" button |
//+-----+
bool CControlsDialog::CreateButton2(void)
{
//--- coordinates
    int x1=INDENT_LEFT+(BUTTON_WIDTH+CONTROLS_GAP_X);
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_button2.Create(m_chart_id,m_name+"Button2",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!m_button2.Text("Button2"))
        return(false);
    if(!Add(m_button2))
        return(false);
//--- succeed
    return(true);
}

//+-----+
//| Create the "Button3" fixed button |
//+-----+
bool CControlsDialog::CreateButton3(void)
{
//--- coordinates
    int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_button3.Create(m_chart_id,m_name+"Button3",m_subwin,x1,y1,x2,y2))
        return(false);

```



```

    if (!m_button3.Text("Locked"))
        return(false);
    if (!Add(m_button3))
        return(false);
    m_button3.Locking(true);
    //--- succeed
    return(true);
}

//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnClickButton1(void)
{
    Comment(__FUNCTION__);
}

//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnClickButton2(void)
{
    Comment(__FUNCTION__);
}

//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnClickButton3(void)
{
    if (m_button3.Pressed())
        Comment(__FUNCTION__+" State of the control: On");
    else
        Comment(__FUNCTION__+" State of the control: Off");
}

//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if (!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}

```



```
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear comments
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id, lparam, dparam, sparam);
}
```


Create

Creates new CButton control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,    // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2        // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool Pressed() const
```

Return Value

"Pressed" property of the control.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool Pressed(  
    const bool pressed // state  
)
```

Parameters

pressed

[in] New control state.

Return Value

true - successful, otherwise - false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool Locking() const
```

Return Value

The value of "Locking" property.

Locking (Set method)

Sets the value of "Locking" property of the control.

```
void Locking(  
    const bool locking    // value  
)
```

Parameters

locking

[in] New value of "Locking" property of the control.

Return Value

None.

OnSetText

The virtual handler of control "SetText" (change of the [OBJPROP_TEXT](#) property) event.

```
virtual bool OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the [OBJPROP_COLOR](#) property) event.

```
virtual bool OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the [OBJPROP_BGCOLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the [OBJPROP_BORDER_COLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the [OBJPROP_FONT](#) property) event.

```
virtual bool OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the [OBJPROP_FONTSIZE](#) property) event.

```
virtual bool OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnMouseDown

The virtual handler of control "MouseDown" event handler.

```
virtual bool OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The virtual handler of control "MouseUp" (left mouse button release) event.

```
virtual bool OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

CEdit

CEdit is class of the simple control based on "Edit" chart object.

Description

CEdit class is intended for creation of controls, in which the user can enter text.

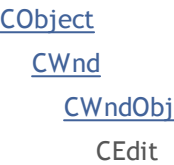
Declaration

```
class CEdit : public CWndObj
```

Title

```
#include <Controls\Edit.mqh>
```

Inheritance hierarchy



Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

Properties	
ReadOnly	Gets/sets the "ReadOnly" property
TextAlign	Gets/sets the "TextAlign" property
Chart object event handlers	
OnObjectEndEdit	The CHARTEVENT_OBJECT_ENDEDIT event handler (virtual)
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnChange	"Change" internal event handler
OnClick	"Click" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with Edit control:

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CEdit"
#include <Controls\Dialog.mqh>
#include <Controls\Edit.mqh>

//+-----+
//| defines |
//+-----+

//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowance)
#define INDENT_TOP            (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM         (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT            (179)     // size by Y coordinate
#define RADIO_HEIGHT           (56)      // size by Y coordinate
#define CHECK_HEIGHT           (93)      // size by Y coordinate

//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CEdit          m_edit;              // CEdit object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
```



```

protected:
    ///--- create dependent controls
    bool      CreateEdit(void);
};
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    ///--- create dependent controls
    if(!CreateEdit())
        return(false);
    ///--- succeed
    return(true);
}
//+-----+
//| Create the display field |
//+-----+
bool CControlsDialog::CreateEdit(void)
{
    ///--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=ClientAreaWidth()-INDENT_RIGHT;
    int y2=y1+EDIT_HEIGHT;
    ///--- create
    if(!m_edit.Create(m_chart_id,m_name+"Edit",m_subwin,x1,y1,x2,y2))
        return(false);
    ///--- allow editing the content
    if(!m_edit.ReadOnly(false))
        return(false);
    if(!Add(m_edit))
        return(false);
    ///--- succeed
    return(true);
}

```



```

    }
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear comments
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates CEdit control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,         // coordinate  
    const int     y1,         // coordinate  
    const int     x2,         // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

ReadOnly (Get Method)

Gets the "ReadOnly" property of the control.

```
bool ReadOnly()
```

Return Value

The value of "ReadOnly" property.

ReadOnly (Set Method)

Sets the value of "ReadOnly" property of the control.

```
bool ReadOnly(  
    const bool flag    // value  
)
```

Parameters

flag

[in] New value of "ReadOnly" property.

Return Value

true - successful, otherwise - false.

TextAlign (Get method)

Gets the value of "TextAlign" property ([text alignment mode](#)) of the control.

```
ENUM_ALIGN_MODE TextAlign() const
```

Return Value

Value of "TextAlign" property of the control.

TextAlign (Set method)

Sets new value of "TextAlign" property ([text alignment mode](#)) of the control.

```
bool TextAlign(  
    ENUM_ALIGN_MODE align    // property value  
)
```

Parameters

align

[in] New value of "TextAlign" property.

Return Value

true - successful, false - cannot change the property.

OnObjectEndEdit

The virtual handler of [CHARTEVENT_OBJECT_ENDEDIT](#) chart object event.

```
virtual bool OnObjectEndEdit ()
```

Return Value

true - event processed, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the [OBJPROP_TEXT](#) property) event.

```
virtual bool OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the [OBJPROP_COLOR](#) property) event.

```
virtual bool OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the [OBJPROP_BGCOLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the [OBJPROP_BORDER_COLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the [OBJPROP_FONT](#) property) event.

```
virtual bool OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the [OBJPROP_FONTSIZE](#) property) event.

```
virtual bool OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnSetZOrder

The virtual handler of control "SetZOrder" (change of the [OBJPROP_ZORDER](#) property) event.

```
virtual bool OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

OnClick

The virtual handler of control "Click" (mouse button click) internal event.

```
virtual bool OnClick()
```

Return Value

true - event processed, otherwise - false.

CPanel

CPanel is a class of the simple control based on "Rectangle label" chart object.

Description

CPanel class is intended to combine the controls with similar functions in the group.

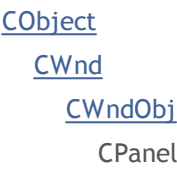
Declaration

```
class CPanel : public CWndObj
```

Title

```
#include <Controls\Panel.mqh>
```

Inheritance hierarchy



Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

Chart object properties	
BorderType	Gets the "BorderType" property of the chart object
Chart object event handlers	
OnSetText	"SetText" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with Rectangle label:

```
//+-----+
//|                                     ControlsPanel.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPanel"
#include <Controls\Dialog.mqh>
```



```

//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowa
#define INDENT_TOP            (11)      // indent from top (with allowa
#define INDENT_RIGHT          (11)      // indent from right (with allowa
#define INDENT_BOTTOM         (11)      // indent from bottom (with allowa
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)     // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)     // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT            (179)    // size by Y coordinate
#define RADIO_HEIGHT           (56)     // size by Y coordinate
#define CHECK_HEIGHT           (93)     // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const

protected:
    //--- create dependent controls
    bool CreatePanel(void);
};
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+

```



```

//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreatePanel())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "CPanel" |
//+-----+
bool CControlsDialog::CreatePanel(void)
{
//--- coordinates
    int x1=20;
    int y1=20;
    int x2=ExtDialog.Width()/3;
    int y2=ExtDialog.Height()/3;
//--- create
    if(!my_white_border.Create(0,ExtDialog.Name()+"MyWhiteBorder",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!my_white_border.ColorBackground(CONTROLS_DIALOG_COLOR_BG))
        return(false);
    if(!my_white_border.ColorBorder(CONTROLS_DIALOG_COLOR_BORDER_LIGHT))
        return(false);
    if(!ExtDialog.Add(my_white_border))
        return(false);
    my_white_border.Alignment(WND_ALIGN_CLIENT,0,0,0,0);
//--- succeed
    return(true);
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//---
CPanel my_white_border; // object CPanel
bool pause=true; // true - pause
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//---
    EventSetTimer(3);

```



```

    pause=true;
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- clear comments
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

//+-----+
//| Timer |
//+-----+
void OnTimer()
{
    pause=!pause;
}

```


Create

Creates new CPanel control.

```
virtual bool Create(  
    const long   chart,      // chart ID  
    const string name,      // name  
    const int    subwin,    // chart subwindow  
    const int    x1,        // coordinate  
    const int    y1,        // coordinate  
    const int    x2,        // coordinate  
    const int    y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

BorderType (Get method)

Gets the "BorderType" property of the chart object.

```
ENUM_BORDER_TYPE BorderType()
```

Return Value

The value of "BorderType" property.

BorderType (Set method)

Sets new value of "BorderType" property of the chart object.

```
bool BorderType (  
    const ENUM_BORDER_TYPE type // value  
)
```

Parameters

type

[in] New value of "BorderType" property.

Return Value

true - successful, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the [OBJPROP_TEXT](#) property) event.

```
virtual bool OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the [OBJPROP_BGCOLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the [OBJPROP_BORDER_COLOR](#) property) event.

```
virtual bool OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

CPicture

CPicture is a class of the simple control based on "Bitmap Label" object.

Description

CPicture class is intended for creation of simple graphic images.

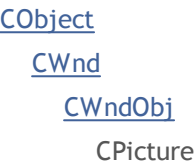
Declaration

```
class CPicture : public CWndObj
```

Title

```
#include <Controls\Picture.mqh>
```

Inheritance hierarchy



Result of the [code](#) provided below:



Class Methods by Groups

Create	
Create	Creates control

Chart object properties	
Border	Gets/sets the border width of the chart object
BmpName	Gets/sets the name of bmp file of the control
Internal events	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)

Methods inherited from class CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Example of creating a panel with Bitmap label:

```
//+-----+
//|                                     ControlsPicture.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPicture"
#include <Controls\Dialog.mqh>
#include <Controls\Picture.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT      (11)      // indent from left (with allowance)
#define INDENT_TOP       (11)      // indent from top (with allowance)
```



```

#define INDENT_RIGHT (11) // indent from right (with allow
#define INDENT_BOTTOM (11) // indent from bottom (with allow
#define CONTROLS_GAP_X (5) // gap by X coordinate
#define CONTROLS_GAP_Y (5) // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH (100) // size by X coordinate
#define BUTTON_HEIGHT (20) // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT (20) // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH (150) // size by X coordinate
#define LIST_HEIGHT (179) // size by Y coordinate
#define RADIO_HEIGHT (56) // size by Y coordinate
#define CHECK_HEIGHT (93) // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CPicture m_picture; // CPicture object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool CreatePicture(void);
    //--- handlers of the dependent controls events
    void OnClickPicture(void);
};
//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_picture,OnClickPicture)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

```



```

//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreatePicture())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "Picture" |
//+-----+
bool CControlsDialog::CreatePicture(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+32;
    int y2=y1+32;
//--- create
    if(!m_picture.Create(m_chart_id,m_name+"Picture",m_subwin,x1,y1,x2,y2))
        return(false);
//--- set the name of bmp files to display the CPicture control
    m_picture.BmpName("\\Images\\euro.bmp");

    if(!Add(m_picture))
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnClickPicture(void)
{
    Comment(__FUNCTION__);
}
//+-----+
//| Global Variables |

```



```

//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
//--- clear comments
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}

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

void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates new CPicture control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int Border() const
```

Return Value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool Border(  
    const int value    // value  
)
```

Parameters

value

[in] New value of the "Border" property.

Return Value

true - successful, otherwise - false.

BmpName (Get method)

Gets the name of bmp file of the control.

```
string BmpName() const
```

Return Value

Name of bmp file of the control.

BmpName (Set method)

Sets the name of bmp file of the control.

```
bool BmpName (  
    const string name // file name  
)
```

Parameters

name

[in] Name of bmp file of the control.

Return Value

true - successful, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool OnMove()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool OnChange()
```

Return Value

true - event processed, otherwise - false.

CScroll

CScroll is a base class for creation of scroll bars.

Description

CScroll is a complex control (with dependent controls), it contains the base functionality for creation of scroll bars. The base class itself is not used as a separate control, two of its heirs ([CScrollV](#) and [CScrollH](#) classes) are used as controls.

Declaration

```
class CScroll : public CWndContainer
```

Title

```
#include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CScroll

Direct descendants

[CScrollH](#), [CScrollV](#)

Class Methods by Groups

Create	
Create	Creates control
Chart object event handlers	
OnEvent	Event handler of all chart events
Properties	
MinPos	Gets/sets the minimal position
MaxPos	Gets/sets the maximal position
CurrPos	Gets/sets the current position
Dependent controls creation	
CreateBack	Creates background button
CreateInc	Creates increment button of the scroll bar
CreateDec	Creates decrement button of the scroll bar

CreateThumb	Creates thumb button (can be dragged) of the scroll bar
Dependent controls event handlers	
OnClickInc	Event handler used for handling increment button events
OnClickDec	Event handler used for handling decrement button events
Internal event handlers	
OnShow	"Create" internal event handler
OnHide	"Hide" internal event handler
OnChangePos	"ChangePosition" internal event handler
Object drag handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

Create

Creates new CScroll control.

```
virtual bool Create(  
    const long   chart,      // chart ID  
    const string name,      // name  
    const int    subwin,    // chart subwindow  
    const int    x1,        // coordinate  
    const int    y1,        // coordinate  
    const int    x2,        // coordinate  
    const int    y2        // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

MinPos (Get method)

Gets the value of "MinPos" (minimal position) of the CScroll control.

```
int MinPos() const
```

Return Value

New value of "MinPos" property.

MinPos (Set method)

Sets the value of "MinPos" (minimal position) of the CScroll control.

```
void MinPos(  
    const int value    // value  
)
```

Parameters

value

[in] New value of "MinPos" property.

Return Value

None.

MaxPos (Get method)

Gets the value of "MaxPos" (maximal position) of the CScroll control.

```
int MaxPos() const
```

Return Value

New value of "MaxPos" property.

MaxPos (Set method)

Sets the value of "MaxPos" (maximal position) of the CScroll control.

```
void MaxPos(  
    const int value    // value  
)
```

Parameters

value

[in] New value of "MaxPos" property.

Return Value

None.

CurrPos (Get method)

Gets the value of "CurrPos" (current position) of the CScroll control.

```
int CurrPos() const
```

Return Value

New value of "CurrPos" property.

CurrPos (Set method)

Sets the value of "CurrPos" (current position) of the CScroll control.

```
void CurrPos(  
    const int value    // value  
)
```

Parameters

value

[in] New value of "CurrPos" property.

Return Value

None.

CreateBack

Creates background button of the CScroll control.

```
virtual bool CreateBack()
```

Return Value

true - successful, otherwise - false.

CreateInc

Creates increment button of CScroll control.

```
virtual bool CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of CScroll control.

```
virtual bool CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of CScroll control.

```
virtual bool CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnClickInc

The virtual handler of the control "ClickInc" (left mouse button click on the increment button) internal event.

```
virtual bool OnClickInc()
```

Return Value

true - event processed, otherwise - false.

OnClickDec

The virtual handler of the control "ClickDec" (left mouse button click on the decrement button) internal event.

```
virtual bool OnClickDec()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of the control "Show" event.

```
virtual bool OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of the control "Hide" internal event.

```
virtual bool OnHide()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool OnChangePos()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool OnThumbDragStart ()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool OnThumbDragProcess(  
    const int x,      // x coordinate  
    const int y      // y coordinate  
)
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int CalcPos(  
    const int coord // coordinate  
)
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CScrollV

CScrollV is a class of the "Vertical scroll bar" complex control.

Description

CScrollV class is intended for creation of vertical scroll bars.

Declaration

```
class CScrollV : public CScroll
```

Title

```
#include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

[CScroll](#)

CScrollV

Result of the [code](#) provided below:



Class Methods by Groups

Dependent controls

CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" internal event handler
OnChangePos	"ChangePosition" internal event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

Methods inherited from class CScroll

[Create](#), [OnEvent](#), [MinPos](#), [MinPos](#), [MaxPos](#), [MaxPos](#), [CurrPos](#), [CurrPos](#)

Example of creating a panel with vertical scrollbar:

```
//+-----+
//|                                     ControlsScrollV.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollV"
```



```

#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>

//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)          // indent from left (with allowa
#define INDENT_TOP            (11)          // indent from top (with allowar
#define INDENT_RIGHT          (11)          // indent from right (with allow
#define INDENT_BOTTOM         (11)          // indent from bottom (with allo
#define CONTROLS_GAP_X        (5)           // gap by X coordinate
#define CONTROLS_GAP_Y        (5)           // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)         // size by X coordinate
#define BUTTON_HEIGHT          (20)          // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)          // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)         // size by X coordinate
#define LIST_HEIGHT            (179)         // size by Y coordinate
#define RADIO_HEIGHT           (56)          // size by Y coordinate
#define CHECK_HEIGHT           (93)          // size by Y coordinate
//+-----+
//| Class CControlsDialog                       |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CScrollV          m_scroll_v;           // CScrollV object

public:
                                CControlsDialog(void);
                                ~CControlsDialog(void);

    //--- create
    virtual bool          Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool          OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool                  CreateScrollV(void);
    //--- handlers of the dependent controls events
    void                  OnScrollInc(void);
    void                  OnScrollDec(void);
};
//+-----+
//| Event Handling                               |
//+-----+

```



```

EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateScrollV())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the CScrollsV object |
//+-----+
bool CControlsDialog::CreateScrollV(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=x1+18;
    int y2=y1+LIST_HEIGHT;
//--- create
    if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollV",m_subwin,x1,y1,x2,y2))
        return(false);
//--- set up the scrollbar
    m_scroll_v.MinPos(0);
//--- set up the scrollbar
    m_scroll_v.MaxPos(10);
    if(!Add(m_scroll_v))
        return(false);
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}

```



```

//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnScrollInc(void)
{
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnScrollDec(void)
{
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear comments
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type

```



```
        const double& dparam, // event parameter of the double type
        const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```


CreateInc

Creates increment button of the control.

```
virtual bool CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of the control.

```
virtual bool CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool OnChangePos()
```

Return Value

true - event processed, otherwise - false.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool OnThumbDragStart ()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool OnThumbDragProcess(  
    const int x,      // x coordinate  
    const int y      // y coordinate  
)
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int CalcPos(  
    const int coord // coordinate  
)
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CScrollH

CScrollH is a class of the "Horizontal scroll bar" complex control.

Description

CScrollH is intended for creation of horizontal scroll bars.

Declaration

```
class CScrollH : public CScroll
```

Title

```
#include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

[CScroll](#)

CScrollH

Result of the [code](#) provided below:



Class Methods by Groups

Dependent controls

CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" internal event handler
OnChangePos	"ChangePosition" internal event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

Methods inherited from class CScroll

[Create](#), [OnEvent](#), [MinPos](#), [MinPos](#), [MaxPos](#), [MaxPos](#), [CurrPos](#), [CurrPos](#)

Example of creating a panel with horizontal scrollbar:

```
//+-----+
//|                                     ControlsScrollH.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollH"
```



```

#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>

//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)          // indent from left (with allowa
#define INDENT_TOP            (11)          // indent from top (with allowar
#define INDENT_RIGHT          (11)          // indent from right (with allow
#define INDENT_BOTTOM         (11)          // indent from bottom (with allo
#define CONTROLS_GAP_X        (5)           // gap by X coordinate
#define CONTROLS_GAP_Y        (5)           // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)         // size by X coordinate
#define BUTTON_HEIGHT          (20)          // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)          // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)         // size by X coordinate
#define LIST_HEIGHT            (179)         // size by Y coordinate
#define RADIO_HEIGHT           (56)          // size by Y coordinate
#define CHECK_HEIGHT           (93)          // size by Y coordinate
//+-----+
//| Class CControlsDialog                       |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CScrollH          m_scroll_v;           // CScrollH object

public:
                                CControlsDialog(void);
                                ~CControlsDialog(void);

    //--- create
    virtual bool         Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool         OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool                 CreateScrollsH(void);
    //--- handlers of the dependent controls events
    void                 OnScrollInc(void);
    void                 OnScrollDec(void);
};
//+-----+
//| Event Handling                               |
//+-----+

```



```

EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateScrollsH())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the CScrollsH object |
//+-----+
bool CControlsDialog::CreateScrollsH(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=x1+3*BUTTON_WIDTH;
    int y2=y1+18;
//--- create
    if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollsH",m_subwin,x1,y1,x2,y2))
        return(false);
//--- set up the scrollbar
    m_scroll_v.MinPos(0);
//--- set up the scrollbar
    m_scroll_v.MaxPos(10);
    if(!Add(m_scroll_v))
        return(false);
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}

```



```

//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnScrollInc(void)
{
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnScrollDec(void)
{
    Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear comments
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type

```



```
const double& dparam, // event parameter of the double type
const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```


CreateInc

Creates increment button of the control.

```
virtual bool CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of the control.

```
virtual bool CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool OnChangePos()
```

Return Value

true - event processed, otherwise - false.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool OnThumbDragStart ()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool OnThumbDragProcess(  
    const int x,      // x coordinate  
    const int y      // y coordinate  
)
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int CalcPos(  
    const int coord // coordinate  
)
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CWndClient

CWndClient is a class of the "Client area" complex control (with dependent controls). It is a base class for creation of scroll bars area.

Description

CWndClient implements the functionality for creation of client area with scroll bars.

Declaration

```
class CWndClient : public CWndContainer
```

Title

```
#include <Controls\WndClient.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CWndClient

Direct descendants

[CCheckGroup](#), [CListView](#), [CRadioGroup](#)

Class Methods by Groups

Create	
Create	Creates control
Chart event handler	
OnEvent	Event handler of all chart events
Properties	
ColorBackground	Sets background color
ColorBorder	Sets border color
BorderType	Sets border type
Settings	
VScrolled	Gets/sets the flag indicating that vertical scroll bar is used
HScrolled	Gets/sets the flag indicating that horizontal scroll bar is used
Dependent controls	

CreateBack	Creates background for scroll bar
CreateScrollV	Creates vertical scroll bar
CreateScrollH	Creates horizontal scroll bar
Internal event handlers	
OnResize	"Resize" internal event handler
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnHScrollShow	"Show" internal event handler (virtual) of HScroll dependent control
OnHScrollHide	"Hide" internal event handler (virtual) of HScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnScrollLineLeft	"ScrollLineLeft" internal event handler (virtual) of HScroll dependent control
OnScrollLineRight	"ScrollLineRight" internal event handler (virtual) of HScroll dependent control
Resize	
Rebound	Sets new parameters of the control using CRect class coordinates

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#), [Save](#), [Load](#)

Create

Creates new CWndClient control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,         // coordinate  
    const int     y1,         // coordinate  
    const int     x2,         // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,      // parameter  
    const double&  dparam,      // parameter  
    const string&  sparam       // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

ColorBackground

Sets background color of the control.

```
bool ColorBackground(  
    const color value    // background color  
)
```

Parameters

value

[in] Background color of the control.

Return Value

true - successful, otherwise - false.

ColorBorder

Set border color of the control.

```
bool ColorBorder(  
    const color value    // color  
)
```

Parameters

value

[in] Border color of the control.

Return Value

true - successful, otherwise - false.

BorderStyle

Sets border type of the control.

```
bool BorderType(  
    const ENUM_BORDER_TYPE type // value  
)
```

Parameters

type

[in] Border type of the control.

Return Value

true - successful, otherwise - false.

VScrolled (Get method)

Gets the flag indicating that vertical scroll bar is used.

```
bool VScrolled()
```

Return Value

true - vertical scroll bar is used, otherwise - false.

VScrolled (Set method)

Sets the flag indicating that vertical scroll bar is used.

```
bool VScrolled(  
    const bool flag    // flag  
)
```

Parameters

flag

[in] Flag.

Return Value

true - successful, otherwise - false.

HScrolled (Get method)

Gets the flag indicating that horizontal scroll bar is used.

```
bool HScrolled()
```

Return Value

true - horizontal scroll bar is used, otherwise - false.

HScrolled (Set method)

Sets the flag indicating that horizontal scroll bar is used.

```
bool HScrolled(  
    const bool flag    // flag  
)
```

Parameters

flag

[in] Flag.

Return Value

true - successful, otherwise - false.

CreateBack

Creates background button of the control.

```
virtual bool CreateBack()
```

Return Value

true - successful, otherwise - false.

CreateScrollV

Creates vertical scroll bar.

```
virtual bool CreateScrollV()
```

Return Value

true - successful, otherwise - false.

CreateScrollH

Creates horizontal scroll bar.

```
virtual bool CreateScrollH()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnVScrollShow

The virtual handler of the VScroll (vertical scroll) dependent control "Show" (vertical scroll bar show) internal event.

```
virtual bool OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnVScrollHide

The virtual handler of the VScroll (vertical scroll) dependent control "Hide" (vertical scroll bar hide) internal event.

```
virtual bool OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHScrollShow

The virtual handler of the HScroll (horizontal scroll) dependent control "Show" (horizontal scroll bar show) internal event.

```
virtual bool OnHScrollShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHScrollHide

The virtual handler of the HScroll (horizontal scroll) dependent control "Hide" (horizontal scroll bar hide) internal event.

```
virtual bool OnHScrollHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineDown

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineDown" (vertical scroll line down) internal event.

```
virtual bool OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineUp

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineUp" (vertical scroll line up) internal event.

```
virtual bool OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineLeft

The virtual handler of the HScroll (horizontal scroll) dependent control "ScrollLineLeft" (horizontal scroll line left) internal event.

```
virtual bool OnScrollLineLeft()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineRight

The virtual handler of the HScroll (horizontal scroll) dependent control "ScrollLineRight" (horizontal scroll line right) internal event.

```
virtual bool OnScrollLineRight()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

ReBound

Sets new parameters of the control using CRect class coordinates.

```
void ReBound(  
    const & CRect rect    // CRect class  
)
```

Return Value

None.

CListView

CListView is a class of the ListView complex control (with dependent controls).

Description

CListView class encapsulates list-control functionality.

Declaration

```
class CListView : public CWndClient
```

Title

```
#include <Controls\ListView.mqh>
```

Inheritance hierarchy

[CObject](#)

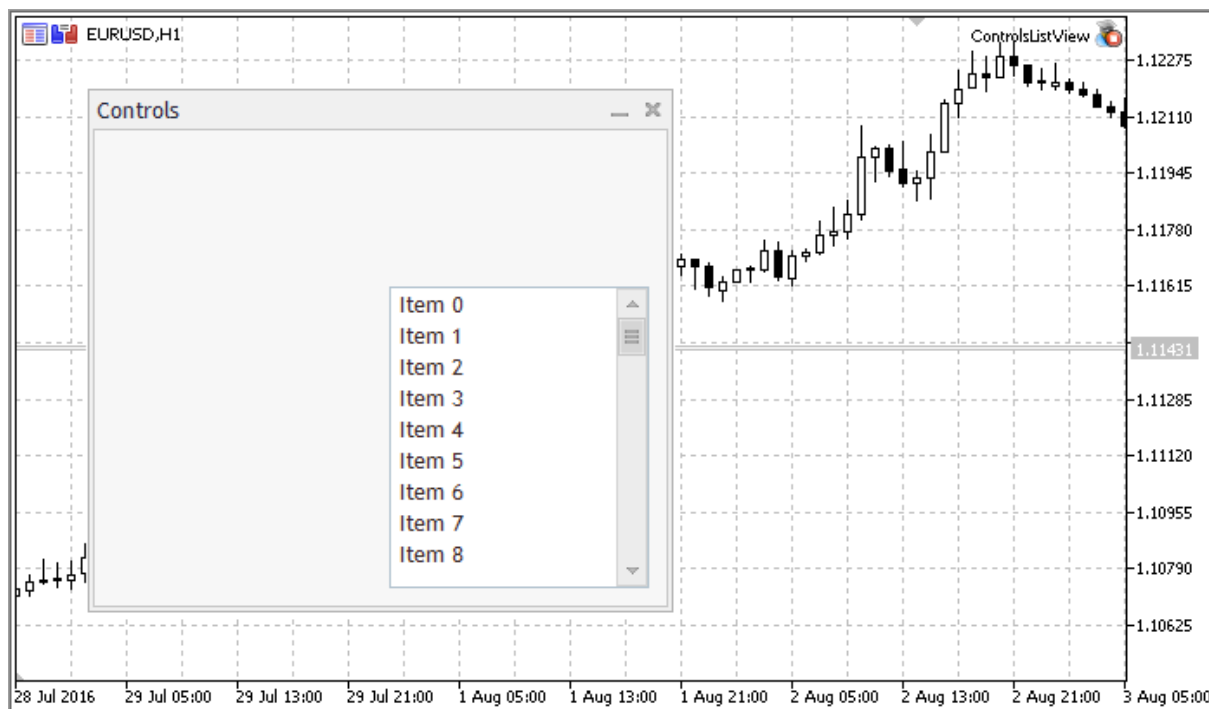
[CWnd](#)

[CWndContainer](#)

[CWndClient](#)

CListView

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Settings	
TotalView	Sets the number of items shown on the control
Add/Delete	
AddItem	Adds an item to control list
Data	
Select	Selects current list element by index
SelectByText	Selects current list element by text
SelectByValue	Selects current list element by value
Read-only data	
Value	Gets the value of the current list element
Dependent controls	
CreateRow	Creates a row of the ListView
Internal event handlers	
OnResize	"Resize" internal event handler (virtual)
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnItemClick	"ItemClick" internal event handler (virtual)
Redraw	
Redraw	Redraws the control
RowState	Sets the state of the specified row
CheckView	Checks the "visibility" of the specified row

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#), [Save](#), [Load](#)

Methods inherited from class CWndClient

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

Example of creating a panel with list view control:

```
//+-----+
//|                                     ControlsListView.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CListView"
#include <Controls\Dialog.mqh>
#include <Controls\ListView.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowa
#define INDENT_TOP            (11)      // indent from top (with allowar
#define INDENT_RIGHT          (11)      // indent from right (with allow
#define INDENT_BOTTOM         (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT            (179)     // size by Y coordinate
#define RADIO_HEIGHT           (56)      // size by Y coordinate
#define CHECK_HEIGHT           (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog |
```



```

//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CListView      m_list_view;          // CListView object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool            CreateListView(void);
    //--- handlers of the dependent controls events
    void            OnChangeListView(void);
};
//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_list_view,OnChangeListView)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateListView())
        return(false);
}

```



```

//--- succeed
    return(true);
}

//+-----+
//| Create the "ListView" element |
//+-----+

bool CControlsDialog::CreateListView(void)
{
//--- coordinates
    int x1=INDENT_LEFT+GROUP_WIDTH+2*CONTROLS_GAP_X;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+2*CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+LIST_HEIGHT-CONTROLS_GAP_Y;
//--- create
    if(!m_list_view.Create(m_chart_id,m_name+"ListView",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_list_view))
        return(false);
//--- fill out with strings
    for(int i=0;i<16;i++)
        if(!m_list_view.AddItem("Item "+IntegerToString(i)))
            return(false);
//--- succeed
    return(true);
}

//+-----+
//| Event handler |
//+-----+

void CControlsDialog::OnChangeListView(void)
{
    Comment(__FUNCTION__+" \" "+m_list_view.Select()+"\");
}

//+-----+
//| Global Variables |
//+-----+

CControlsDialog ExtDialog;

//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed

```



```
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear comments
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```


Create

Creates CListView control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

TotalView

Sets the number of items shown on the control.

```
bool TotalView(  
    const int value    // number of items shown  
)
```

Parameters

value

[in] The number of items shown on the control.

Return Value

true - successful, otherwise - false.

Note

The number of shown items can be specified only at once.

AddItem

Adds an item to the control list.

```
bool AddItem(  
    const string item,      // text  
    const long  value      // value  
)
```

Parameters

item

[in] Text.

value

[in] Value.

Return Value

true - successful, otherwise - false.

Select

Selects current list element by index.

```
bool Select(  
    const int index    // index  
)
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

SelectByText

Selects the current list element by text.

```
bool SelectByText(  
    const string text    // text  
)
```

Parameters

text

[in] Element text.

Return Value

true - successful, otherwise - false.

SelectByValue

Selects the current list element by value.

```
bool SelectByValue(  
    const long value    // value  
)
```

Parameters

value

[in] Value.

Return Value

true - successful, otherwise - false.

Value

Gets the value of the current list element.

```
long Value()
```

Return Value

The value of the current list element.

CreateRow

Creates a row of the CListView control.

```
bool CreateRow(  
    const int index    // index  
)
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool OnResize()
```

Return Value

true - event processed, otherwise - false.

OnVScrollShow

The virtual handler of the VScroll (vertical scroll) dependent control "VScrollShow" (vertical scroll bar show) internal event.

```
virtual bool OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the VScroll (vertical scroll) dependent control "VScrollHide" (vertical scroll bar hide) internal event.

```
virtual bool OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineDown" (vertical scroll line down) internal event.

```
virtual bool OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineUp" (vertical scroll line up) internal event.

```
virtual bool OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnItemClick

The virtual handler of "ItemClick" (mouse button click) internal event on a specified row of CListView control.

```
virtual bool OnItemClick()  
    const int    index    // index  
    )
```

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Changes the state of the specified row of the CListView control.

```
bool RowState(  
    const int    index        // index  
    const bool   select       // state  
)
```

Parameters

index

[in] Row index.

select

[in] Row state.

Return Value

true - successful, otherwise - false.

CheckView

Checks the "visibility" of the specified row of the CListView control.

```
bool CheckView()
```

Return Value

true - selected row is visible, otherwise - false.

CComboBox

CComboBox is a class of the ComboBox complex control (with dependent controls).

Description

ComboBox consists of a list box, combined with a static control, intended for selection. The list-box portion of the control may be dropped down when a user selects the drop-down arrow next to the control

Declaration

```
class CComboBox : public CWndContainer
```

Title

```
#include <Controls\ComboBox.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CComboBox

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds an item to control
Settings	
ListViewItems	Sets the number of items shown on the control
Data	
Select	Selects the current list element by index
SelectByText	Selects the current list element by text
SelectByValue	Selects the current list element by value
Read-only data	
Value	Gets the value of the current list element
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateButton	Creates dependent control (button)
CreateList	Creates dependent control (list view)
Dependent controls event handlers	
OnClickEdit	"ClickEdit" internal event handler (virtual)
OnClickButton	"ClickButton" internal event handler (virtual)
OnChangeList	"ChangeList" internal event handler (virtual)
Show/Hide the drop-down list	
ListShow	Shows the items list
ListHide	Hides the items list

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), Size, Size, Size, [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), BringToTop

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Hide](#)

Example of creating a panel with Combobox control:

```
//+-----+
//|                                     ControlsComboBox.mqh |
//|                                     Copyright 2015, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+

#property copyright "Copyright 2015, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CComboBox"
#include <Controls\Dialog.mqh>
#include <Controls\ComboBox.mqh>

//+-----+
//| defines |
//+-----+

//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowance)
#define INDENT_TOP            (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM         (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT            (179)     // size by Y coordinate
#define RADIO_HEIGHT           (56)      // size by Y coordinate
#define CHECK_HEIGHT           (93)      // size by Y coordinate

//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CComboBox      m_combo_box;;        // CComboBox object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
```



```

virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool          CreateComboBox(void) ;
    //--- handlers of the dependent controls events
    void          OnChangeComboBox(void) ;
};
//+-----+
//| Event Handling                                     |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_combo_box,OnChangeComboBox)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor                                     |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor                                     |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create                                     |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateComboBox())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "ComboBox" element                                     |
//+-----+
bool CControlsDialog::CreateComboBox(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+

```



```

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+EDIT_HEIGHT;
    ///--- create
    if(!m_combo_box.Create(m_chart_id,m_name+"ComboBox",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_combo_box))
        return(false);
    ///--- fill out with strings
    for(int i=0;i<16;i++)
        if(!m_combo_box.ItemAdd("Item "+IntegerToString(i)))
            return(false);
    ///--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnChangeComboBox(void)
{
    Comment(__FUNCTION__+" \" "+m_combo_box.Select()+"\");
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    ///--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    ///--- run application
    ExtDialog.Run();
    ///--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    ///---
    Comment("");
    ///--- destroy dialog
    ExtDialog.Destroy(reason);
}

```



```
    }  
    //+-----+  
    //| Expert chart event function |  
    //+-----+  
    void OnChartEvent(const int id,          // event ID  
                      const long& lparam,    // event parameter of the long type  
                      const double& dparam,  // event parameter of the double type  
                      const string& sparam) // event parameter of the string type  
        ExtDialog.ChartEvent(id,lparam,dparam,sparam);  
    }
```


Create

Creates CComboBox control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chat, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item to the control.

```
bool AddItem(  
    const string item,      // text  
    const long  value      // value  
)
```

Parameters

item

[in] Text.

value=0

[in] Value.

Return Value

true - successful, otherwise - false.

ListViewItems

Sets the number of list elements of the CComboBox control.

```
void ListViewItems(  
    const int    value    // number of elements  
)
```

Parameters

value

[in] Number of the drop-down list elements.

Return Value

true - successful, otherwise - false.

Select

Selects the current list element by index.

```
bool Select(  
    const int index    // index  
)
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

SelectByText

Selects the current list element by a specified text.

```
bool SelectByText(  
    const string text    // text  
)
```

Parameters

text

[in] Text of the item.

Return Value

true - successful, otherwise - false.

SelectByValue

Selects the current list element by a specified value.

```
bool SelectByValue(  
    const long value    // value  
)
```

Parameters

value

[in] Value.

Return Value

true - successful, otherwise - false.

Value

Gets the value of the current list element.

```
long Value()
```

Return Value

The value of the current list element.

CreateEdit

Creates dependent control (edit) of the control.

```
virtual bool CreateEdit()
```

Return Value

true - successful, otherwise - false.

CreateButton

Creates dependent control (button).

```
virtual bool CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateList

Creates dependent control (list view).

```
virtual bool CreateList()
```

Return Value

true - successful, otherwise - false.

OnClickEdit

The virtual handler of the control "ClickEdit" (mouse click on the edit) internal event.

```
virtual bool OnClickEdit()
```

Return Value

true - event processed, otherwise - false.

OnClickButton

The virtual handler of the "ClickButton" (mouse click on the button) internal event.

```
virtual bool OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnChangeList

The virtual handler of the "ChangeList" (change of the list) internal event.

```
virtual bool OnChangeList()
```

Return Value

true - event processed, otherwise - false.

ListShow

Shows the drop-down list of the control.

```
virtual bool ListShow()
```

Return Value

true - successful, otherwise - false.

ListHide

Hides the drop-down list of the control.

```
virtual bool ListHide()
```

Return Value

true - successful, otherwise - false.

CCheckBox

CCheckBox is class of the CheckBox complex control.

Description

CCheckBox control displays a check box that allows a user to select a true or false condition at a mouse click.

Declaration

```
class CCheckBox : public CWndContainer
```

Title

```
#include <Controls\CheckBox.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CCheckBox

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/sets text label associated with the control
Color	Gets/sets color of text label associated with the control
State	
Checked	Gets/sets a value indicating whether the control is checked
Data	
Value	Gets/sets the value associated with the control
Dependent controls	
CreateButton	Creates dependent control (button)
CreateLabel	Creates dependent control (label)
Dependent controls event handlers	
ClickButton	"ClickButton" internal event handler (virtual)
ClickLabel	"ClickLabel" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Example of creating a panel with Checkbox control:

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



```

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckBox"
#include <Controls\Dialog.mqh>
#include <Controls\CheckBox.mqh>

//+-----+
//| defines                                     |
//+-----+

//--- indents and gaps
#define INDENT_LEFT           (11)           // indent from left (with allowance)
#define INDENT_TOP            (11)           // indent from top (with allowance)
#define INDENT_RIGHT          (11)           // indent from right (with allowance)
#define INDENT_BOTTOM         (11)           // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)            // gap by X coordinate
#define CONTROLS_GAP_Y        (5)            // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT            (20)           // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH            (150)          // size by X coordinate
#define LIST_HEIGHT            (179)          // size by Y coordinate
#define RADIO_HEIGHT           (56)           // size by Y coordinate
#define CHECK_HEIGHT           (93)           // size by Y coordinate

//+-----+
//| Class CControlsDialog                       |
//| Usage: main dialog of the Controls application |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CCheckBox      m_check_box1;               // CCheckBox object
    CCheckBox      m_check_box2;               // CCheckBox object
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool   Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool   OnEvent(const int id,const long &lparam,const double &dparam,const
protected:
    //--- create dependent controls
    bool           CreateCheckBox1(void);
    bool           CreateCheckBox2(void);
    //--- handlers of the dependent controls events
    void           OnChangeCheckBox1(void);
    void           OnChangeCheckBox2(void);

```



```

};
//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_box1,OnChangeCheckBox1)
ON_EVENT(ON_CHANGE,m_check_box2,OnChangeCheckBox2)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateCheckBox1())
        return(false);
    if(!CreateCheckBox2())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "CheckBox" element |
//+-----+
bool CControlsDialog::CreateCheckBox1(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (RADIO_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+BUTTON_HEIGHT;

```



```

//--- create
if(!m_check_box1.Create(m_chart_id,m_name+"CheckBox1",m_subwin,x1,y1,x2,y2))
    return(false);
if(!m_check_box1.Text("CheckBox1"))
    return(false);
if(!m_check_box1.Color(clrBlue))
    return(false);
if(!Add(m_check_box1))
    return(false);
//--- succeed
return(true);
}
//+-----+
//| Create the "CheckBox" element |
//+-----+
bool CControlsDialog::CreateCheckBox2(void)
{
//--- coordinates
int x1=INDENT_LEFT+GROUP_WIDTH+CONTROLS_GAP_X;
int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
    (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
    (EDIT_HEIGHT+CONTROLS_GAP_Y)+
    (EDIT_HEIGHT+CONTROLS_GAP_Y)+
    (RADIO_HEIGHT+CONTROLS_GAP_Y);
int x2=x1+GROUP_WIDTH;
int y2=y1+BUTTON_HEIGHT;
//--- create
if(!m_check_box2.Create(m_chart_id,m_name+"CheckBox2",m_subwin,x1,y1,x2,y2))
    return(false);
if(!m_check_box2.Text("CheckBox2"))
    return(false);
if(!m_check_box2.Color(clrBlue))
    return(false);
if(!Add(m_check_box2))
    return(false);
m_check_box2.Checked(true);
Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
//--- succeed
return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnChangeCheckBox1(void)
{
    Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box1.Checked()));
}
//+-----+
//| Event handler |

```



```

//+-----+
void CControlsDialog::OnChangeCheckBox2(void)
{
    Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //---
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates CCheckBox control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,    // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2        // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets text of the label associated with the control.

```
string Text()
```

Return Value

Text of the label.

Text (Set method)

Sets text of the label associated with the control.

```
bool Text(  
    const string value    // text  
)
```

Parameters

value

[in] New text of the label.

Return Value

true - successful, otherwise - false.

Note

Label text is specified by setting [OBJPROP_TEXT](#) (text) of a chart object.

Color (Get method)

Gets color of the label associated with the control.

```
color Color() const
```

Return Value

Label color.

Color (Set method)

Sets color of the label associated with the control.

```
bool Color(  
    const color value    // color  
)
```

Parameters

value

[in] New label color.

Return Value

true - successful, otherwise - false.

Note

Label color is specified by setting [OBJPROP_COLOR](#) (color) of a chart object.

Checked (Get method)

Gets state of the control.

```
bool Checked() const
```

Return Value

State of the control.

Checked (Set method)

Sets state of the control.

```
bool Checked(  
    const bool flag    // state  
)
```

Parameters

flag

[in] New state.

Return Value

true - successful, otherwise - false.

Value (Get method)

Gets the value associated with the control.

```
int Value() const
```

Return Value

The value associated with the control.

Value (Set method)

Sets the value associated with the control.

```
void Value(  
    const int value    // value  
)
```

Parameters

value

[in] New value.

Return Value

None.

CreateButton

Creates dependent control (button).

```
virtual bool CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateLabel

Creates dependent control (label).

```
virtual bool CreateLabel()
```

Return Value

true - successful, otherwise - false.

OnClickButton

The virtual handler of the control "ClickButton" (mouse click on the button) internal event.

```
virtual bool OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnClickLabel

The virtual handler of the control "ClickLabel" (mouse click on the label) internal event.

```
virtual bool OnClickLabel ()
```

Return Value

true - event processed, otherwise - false.

CCheckGroup

CCheckGroup is a class of the CheckGroup complex control (with dependent controls).

Description

CCheckGroup provides the possibility for creation of controls, which allow to display and edit flags.

Declaration

```
class CCheckGroup : public CWndClient
```

Title

```
#include <Controls\CheckGroup.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

[CWndClient](#)

CCheckGroup

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value associated with the control
Dependent controls	
CreateButton	Creates new CCheckBox item at a specified index
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" internal event handler (virtual) of VScroll dependent control
Redraw	
Redraw	Redraws the group
RowState	Sets the state of the specified item of the group

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), Size, Size, Size, [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), BringToTop

Methods inherited from class CWndContainer

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#)

Methods inherited from class CWndClient

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

Example of creating a panel with Checkbox group control:

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckGroup"
#include <Controls\Dialog.mqh>
#include <Controls\CheckGroup.mqh>

//+-----+
//| defines                                     |
//+-----+

//--- indents and gaps
#define INDENT_LEFT           (11)           // indent from left (with allowance)
#define INDENT_TOP            (11)           // indent from top (with allowance)
#define INDENT_RIGHT          (11)           // indent from right (with allowance)
#define INDENT_BOTTOM         (11)           // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)           // gap by X coordinate
#define CONTROLS_GAP_Y        (5)           // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT            (20)          // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH            (150)         // size by X coordinate
#define LIST_HEIGHT            (179)         // size by Y coordinate
#define RADIO_HEIGHT           (56)          // size by Y coordinate
#define CHECK_HEIGHT           (93)          // size by Y coordinate

//+-----+
//| Class CControlsDialog                                     |
//| Usage: main dialog of the Controls application         |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CCheckGroup      m_check_group;           // CCheckGroup object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create
```



```

    virtual bool      Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool              CreateCheckGroup(void) ;
    //--- handlers of the dependent controls events
    void              OnChangeCheckGroup(void) ;
};
//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_group,OnChangeCheckGroup)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateCheckGroup())
        return(false);
    //--- succeed
    return(true);
}
//+-----+
//| Create the "CheckGroup" element |
//+-----+
bool CControlsDialog::CreateCheckGroup(void)
{
    //--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+

```



```

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (RADIO_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+CHECK_HEIGHT;
//--- create
    if(!m_check_group.Create(m_chart_id,m_name+"CheckGroup",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_check_group))
        return(false);
//--- fill out with strings
    for(int i=0;i<5;i++)
        if(!m_check_group.AddItem("Item "+IntegerToString(i),1<<i))
            return(false);
    m_check_group.Check(0,1<<0);
    m_check_group.Check(2,1<<2);
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnChangeCheckGroup(void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(ChartID(),"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)

```



```
{
//---
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```


Create

Creates new CCheckGroup control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item (row) to the control.

```
bool AddItem(  
    const string item,      // text  
    const long  value      // value  
)
```

Parameters

item

[in] Added control label text.

value=0

[in] Value associated with the added control.

Return Value

true - successful, otherwise - false.

Value

Gets the value associated with the control.

```
long Value()
```

Return Value

The value associated with the control.

Note

The value depends on state of all items of the CCheckGroup.

CreateButton

Creates new CCheckBox class instance at a specified index.

```
bool CreateButton(  
    int index    // index  
)
```

Parameters

index

[in] Index of the new item in the CCheckGroup.

Return Value

true - successful, otherwise - false.

OnVScrollShow

The virtual handler of the dependent VScroll (vertical scroll) control "Show" internal event.

```
virtual bool OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the dependent VScroll (vertical scroll) control "Hide" internal event.

```
virtual bool OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineDown" internal event.

```
virtual bool OnScrollLineDown ()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineUp" internal event.

```
virtual bool OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnChangeItem

The virtual handler of the control "ChangeItem" (item change) internal event.

```
virtual bool OnChangeItem(  
    const int index    // index  
)
```

Parameters

index

[in] Index of the changed item.

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Sets the state of the specified item.

```
bool RowState(  
    const int    index,      // item index  
    const bool   select      // state  
)
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Return Value

true - successful, otherwise - false.

CRadioButton

CRadioButton is a class of RadioButton complex control.

Description

CRadioButton itself is not used, it used for creation of [CRadioGroup](#) items.

Declaration

```
class CRadioButton : public CWndContainer
```

Title

```
#include <Controls\RadioButton.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CRadioButton

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Properties	
Text	Gets/sets the text label associated with the control
Color	Gets/sets the color of text label associated with the control
State	
State	Gets/sets the state
Dependent controls	
CreateButton	Creates button
CreateLabel	Creates label
Dependent controls event handlers	
OnClickButton	"ClickButton" internal event handler (virtual)
OnClickLabel	"ClickLabel" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

Create

Creates new CRadioButton control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,      // parameter  
    const double&  dparam,      // parameter  
    const string&  sparam       // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets text of the label associated with the control.

```
string Text()
```

Return Value

Text of the label.

Text (Set method)

Sets text of the label.

```
bool Text(  
    const string value    // text  
)
```

Parameters

value

[in] New text of the label.

Return Value

true - successful, otherwise - false.

Color (Get method)

Gets color of the label associated with the control.

```
color Color() const
```

Return Value

Label color.

Color (Set method)

Sets color of the label associated with the control.

```
bool Color(  
    const color value    // color  
)
```

Parameters

value

[in] New label color.

Return Value

true - successful, otherwise - false.

State (Get method)

Gets the button state.

```
bool State() const
```

Return Value

Button state.

State (Set method)

Sets the button state.

```
bool State(  
    const bool flag    // flag  
)
```

Parameters

flag

[in] New button state.

Return Value

true - successful, otherwise - false.

CreateButton

Creates button.

```
virtual bool CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateLabel

Creates label.

```
virtual bool CreateLabel()
```

Return Value

true - successful, otherwise - false.

OnClickButton

The virtual handler of the control "ClickButton" (button click) internal event.

```
virtual bool OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnClickLabel

The virtual handler of the control "ClickLabel" (click on the label) internal event.

```
virtual bool OnClickLabel ()
```

Return Value

true - event processed, otherwise - false.

CRadioGroup

CRadioGroup is a class of RadioGroup complex control (with dependent controls).

Description

CRadioGroup enables creation of a control allowing the display and editing the enumerable type field.

Declaration

```
class CRadioGroup : public CWndClient
```

Title

```
#include <Controls\RadioGroup.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

[CWndClient](#)

CRadioGroup

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value associated with the control
Dependent controls	
CreateButton	Creates new CRadioButton item at a specified index
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" internal event handler (virtual)
Redraw	
Redraw	Redraws the group of items
RowState	Sets the state of the specified item
Select	Selects the current item

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), Size, Size, Size, [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), BringToTop

Methods inherited from class CWndContainer

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#)

Methods inherited from class CWndClient

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

Example of creating a panel with group of radio buttons:

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CRadioGroup"
#include <Controls\Dialog.mqh>
#include <Controls\RadioGroup.mqh>

//+-----+
//| defines                                     |
//+-----+

//--- indents and gaps
#define INDENT_LEFT           (11)          // indent from left (with allowance)
#define INDENT_TOP            (11)          // indent from top (with allowance)
#define INDENT_RIGHT          (11)          // indent from right (with allowance)
#define INDENT_BOTTOM         (11)          // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)           // gap by X coordinate
#define CONTROLS_GAP_Y        (5)           // gap by Y coordinate

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

//--- for the indication area
#define EDIT_HEIGHT            (20)          // size by Y coordinate

//--- for group controls
#define GROUP_WIDTH            (150)         // size by X coordinate
#define LIST_HEIGHT            (179)         // size by Y coordinate
#define RADIO_HEIGHT           (56)          // size by Y coordinate
#define CHECK_HEIGHT           (93)          // size by Y coordinate

//+-----+
//| Class CControlsDialog                                     |
//| Usage: main dialog of the Controls application           |
//+-----+

class CControlsDialog : public CAppDialog
{
private:
    CRadioGroup      m_radio_group;          // CRadioGroup object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create
```



```

virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool          CreateRadioGroup(void);
    //--- handlers of the dependent controls events
    void          OnChangeRadioGroup(void);
};
//+-----+
//| Event Handling                                     |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_radio_group,OnChangeRadioGroup)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor                                       |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor                                       |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create                                           |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateRadioGroup())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "RadioGroup" element                 |
//+-----+
bool CControlsDialog::CreateRadioGroup(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+

```



```

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y)+
        (EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+RADIO_HEIGHT;
    ///--- create
    if(!m_radio_group.Create(m_chart_id,m_name+"RadioGroup",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_radio_group))
        return(false);
    ///--- fill out with strings
    for(int i=0;i<3;i++)
        if(!m_radio_group.AddItem("Item "+IntegerToString(i),1<<i))
            return(false);
    m_radio_group.Value(1<<2);
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
    ///--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnChangeRadioGroup(void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    ///--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    ///--- run application
    ExtDialog.Run();
    ///--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    ///--- clear comments

```



```
    Comment("");  
    //--- destroy dialog  
    ExtDialog.Destroy(reason);  
}  
//+-----+  
//| Expert chart event function |  
//+-----+  
void OnChartEvent(const int id,          // event ID  
                  const long& lparam,    // event parameter of the long type  
                  const double& dparam, // event parameter of the double type  
                  const string& sparam) // event parameter of the string type  
{  
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);  
}
```


Create

Creates new CRadioGroup control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item to the control.

```
bool AddItem(  
    const string item,      // text  
    const long  value=0    // value  
)
```

Parameters

item

[in] Added control label text.

value=0

[in] Value associated with the added control.

Return Value

true - successful, otherwise - false.

Value

Gets the value associated with the control.

```
long Value()
```

Return Value

The value associated with the control.

Note

The value depends on state of all CRadioButton items of the CRadioGroup control.

CreateButton

Creates new CRadioButton class instance at a specified index.

```
bool CreateButton(  
    const int index    // index  
)
```

Parameters

index

[in] Index of the new item in the CRadioGroup.

Return Value

true - successful, otherwise - false.

OnVScrollShow

The virtual handler of the dependent VScroll (vertical scroll) control "Show" internal event.

```
virtual bool OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the dependent VScroll (vertical scroll) control "Hide" internal event.

```
virtual bool OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineDown" internal event.

```
virtual bool OnScrollLineDown ()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineUp" internal event.

```
virtual bool OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnChangeItem

The virtual handler of the control "Changeltem" (item change) event.

```
virtual bool OnChangeItem(  
    const int index    // index  
)
```

Parameters

index

[in] Index of the changed item.

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Changes the state of an item.

```
bool RowState(  
    const int    index,      // index  
    const bool   select      // state  
)
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Return Value

true - successful, otherwise - false.

Select

Selects the current item.

```
void Select(  
    const int index    // index  
)
```

Parameters

index

[in] Item index to select.

Return Value

None.

CSpinEdit

CSpinEdit is a class of SpinEdit complex control (with dependent controls).

Description

CSpinEdit class is intended for creation of a control, which allows editing a value of integer type with a specified step and within specified limitations.

Declaration

```
class CSpinEdit : public CWndContainer
```

Title

```
#include <Controls\SpinEdit.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CSpinEdit

Result of the [code](#) provided below:



Class Methods by Groups

Create

Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
MinValue	Gets/sets the minimal allowed value
MaxValue	Gets/sets the maximal allowed value
State	
Value	Gets/sets the current value
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateInc	Creates dependent control (increment button)
CreateDec	Creates dependent control (decrement button)
Dependent controls event handlers	
OnClickInc	"ClickInc" internal event handler (virtual)
OnClickDec	"ClickDec" internal event handler (virtual)
Internal event handlers	
OnChangeValue	"ChangeValue" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), Size, Size, Size, [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), BringToTop

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Example of creating a panel with spin edit control:

```
//+-----+
//|                                     ControlsSpinEdit.mq5 |
//|                                     Copyright 2017, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
```



```

#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CSpinEdit"
#include <Controls\Dialog.mqh>
#include <Controls\SpinEdit.mqh>

//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT           (11)      // indent from left (with allowa
#define INDENT_TOP            (11)      // indent from top (with allowa
#define INDENT_RIGHT          (11)      // indent from right (with allowa
#define INDENT_BOTTOM         (11)      // indent from bottom (with allowa
#define CONTROLS_GAP_X       (5)        // gap by X coordinate
#define CONTROLS_GAP_Y       (5)        // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT         (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT           (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH           (150)     // size by X coordinate
#define LIST_HEIGHT           (179)     // size by Y coordinate
#define RADIO_HEIGHT          (56)      // size by Y coordinate
#define CHECK_HEIGHT          (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog                       |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CSpinEdit      m_spin_edit;          // CSpinEdit object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool            CreateSpinEdit(void);
    //--- handlers of the dependent controls events
    void            OnChangeSpinEdit(void);
};
//+-----+

```



```

//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
    ON_EVENT(ON_CHANGE,m_spin_edit,OnChangeSpinEdit)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateSpinEdit())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "SpinEdit" element |
//+-----+
bool CControlsDialog::CreateSpinEdit(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+(BUTTON_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+EDIT_HEIGHT;
//--- create
    if(!m_spin_edit.Create(m_chart_id,m_name+"SpinEdit",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_spin_edit))
        return(false);
    m_spin_edit.MinValue(10);
    m_spin_edit.MaxValue(100);
    m_spin_edit.Value(50);
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
}

```



```

//--- succeed
    return(true);
}
//+-----+
//| Event handler |
//+-----+
void CControlsDialog::OnChangeSpinEdit(void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
}
//+-----+
//| Global Variables |
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- clear comments
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```


Create

Creates new CSpinEdit control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,     // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2         // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

MinValue (Get method)

Gets the value of "MinValue" property (minimal value) of the control.

```
int MinValue() const
```

Return Value

The value of "MinValue" property.

MinValue (Set method)

Sets the value of "MinValue" property (minimal value) of the control.

```
void MinValue(  
    const int value    // value  
)
```

Parameters

value

[in] New value of "MinValue" property.

Return Value

None.

MaxValue (Get method)

Gets the value of "MaxValue" property (maximal value) of the control.

```
int MaxValue() const
```

Return Value

The value of "MaxValue" property.

MaxValue (Set method)

Sets the value of "MaxValue" property (maximal value) of the control.

```
void MaxValue(  
    const int value    // value  
)
```

Parameters

value

[in] New value of "MaxValue" property.

Return Value

None.

Value (Get method)

Gets the "Value" property (current value) of the control.

```
int Value() const
```

Return Value

The "Value" property.

Value (Set method)

Sets the "Value" property (current value) of the control.

```
void Value(  
    const int value    // value  
)
```

Parameters

value

[in] New "Value" property.

Return Value

None.

CreateEdit

Creates dependent control (CEdit).

```
virtual bool CreateEdit()
```

Return Value

true - successful, otherwise - false.

CreateInc

Creates dependent control (increment button).

```
virtual bool CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates dependent control (decrement button).

```
virtual bool CreateDec()
```

Return Value

true - successful, otherwise - false.

OnClickInc

The virtual handler of the control "ClickInc" (mouse click on the increment button) internal event.

```
virtual bool OnClickInc()
```

Return Value

true - event processed, otherwise - false.

OnClickDec

The virtual handler of the control "ClickDec" (mouse click on the decrement button) internal event.

```
virtual bool OnClickDec()
```

Return Value

true - event processed, otherwise - false.

OnChangeValue

The virtual handler of the control "ChangeValue" (changing the current value) internal event.

```
virtual bool OnChangeValue()
```

Return Value

true - event processed, otherwise - false.

CDialog

CDialog is class of the Dialog complex control.

Description

CDialog class is intended to combine the controls with different functions in the group.

Declaration

```
class CDialog : public CWndContainer
```

Title

```
#include <Controls\Dialog.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

CDialog

Direct descendants

[CAppDialog](#)

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Caption	Gets/sets the value of the "Caption" property
Add	
Add	Adds control to the client area
Dependent controls	
CreateWhiteBorder	Creates dependent control (white border)
CreateBackground	Creates dependent control (background)
CreateCaption	Creates dependent control (caption)
CreateButtonClose	Creates dependent control (close button)
CreateClientArea	Creates dependent control (client area)

Dependent controls event handlers	
OnClickCaption	"ClickCaption" internal event handler
OnClickButtonClose	"ClickButtonClose" internal event handler
Access to client area properties	
ClientAreaVisible	Sets a value indicating whether the client area is visible
ClientAreaLeft	Gets X coordinate of the upper-left corner of the control client area
ClientAreaTop	Gets Y coordinate of the upper-left corner of the control client area
ClientAreaRight	Gets X coordinate of the lower-right corner of the control client area
ClientAreaBottom	Gets Y coordinate of the lower-right corner of the control client area
ClientAreaWidth	Gets the client area width
ClientAreaHeight	Gets the client area height
Drag event handlers	
OnDialogDragStart	"DialogDragStart" event handler (virtual)
OnDialogDragProcess	"DialogDragProcess" event handler (virtual)
OnDialogDragEnd	"DialogDragEnd" event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Create

Creates new CDialog control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,    // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2        // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

Caption (Get method)

Gets the "Caption" property of the CDialog control.

```
string MinValue() const
```

Return Value

The "Caption" property.

Caption (Set method)

Sets the "Caption" property of the CDialog control.

```
bool Caption(  
    const string text    // text  
)
```

Parameters

text

[in] New value of "Caption" property.

Return Value

true - successful, otherwise - false.

Add

Adds control to the client area by pointer.

```
bool Add(  
    CWnd *control,           // pointer  
)
```

Parameters

control

[in] Pointer to control.

Return Value

true - successful, otherwise - false.

Add

Adds control to the client area by reference.

```
bool Add(  
    CWnd &control,           // reference  
)
```

Parameters

control

[in] Reference to control.

Return Value

true - successful, otherwise - false.

CreateWhiteBorder

Creates dependent control (white border).

```
virtual bool CreateWhiteBorder()
```

Return Value

true - successful, otherwise - false.

CreateBackground

Creates dependent control (background).

```
virtual bool CreateBackground()
```

Return Value

true - successful, otherwise - false.

CreateCaption

Creates dependent control (caption).

```
virtual bool CreateCaption()
```

Return Value

true - successful, otherwise - false.

CreateButtonClose

Creates dependent control (close button)

```
virtual bool CreateButtonClose()
```

Return Value

true - successful, otherwise - false.

CreateClientArea

Creates dependent control (client area).

```
virtual bool CreateClientArea ()
```

Return Value

true - successful, otherwise - false.

OnClickCaption

The virtual handler of the control "ClickCaption" internal event.

```
virtual bool OnClickCaption()
```

Return Value

true - successful, otherwise - false.

OnClickButtonClose

The virtual handler of the control "ClickButtonClose" internal event.

```
virtual bool OnClickButtonClose()
```

Return Value

true - successful, otherwise - false.

ClientAreaVisible

Sets a flag indicating whether the client area is visible.

```
bool ClientAreaVisible(  
    const bool visible    // visibility flag  
)
```

Parameters

visible

[in] Visibility flag.

Return Value

true - successful, otherwise - false.

ClientAreaLeft

Gets X coordinate of the upper-left corner of the control client area.

```
int ClientAreaLeft()
```

Return Value

The X coordinate of the upper-left corner of the control client area.

ClientAreaTop

Gets Y coordinate of the upper-left corner of the control client area.

```
int ClientAreaTop()
```

Return Value

The Y coordinate of the upper-left corner of the control client area.

ClientAreaRight

Gets X coordinate of the lower-right corner of the control client area.

```
int ClientAreaTop()
```

Return Value

The X coordinate of the lower-right corner of the control client area.

ClientAreaBottom

Gets Y coordinate of the lower-right corner of the control client area.

```
int ClientAreaBottom()
```

Return Value

The Y coordinate of the lower-right corner of the control client area.

ClientAreaWidth

Gets the width of the control client area.

```
int ClientAreaWidth()
```

Return Value

The width of the client area.

ClientAreaHeight

Gets the height of the control client area.

```
int ClientAreaHeight ()
```

Return Value

The height of the control client area.

OnDialogDragStart

The virtual handler of the control "DialogDragStart" event.

```
virtual bool OnDialogDragStart ()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragStart" event occurs at start of the drag of the control.

OnDialogDragProcess

The virtual handler of the control "DialogDragProcess" event.

```
virtual bool OnDialogDragProcess()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragProcess" event occurs when the control is dragged.

OnDialogDragEnd

The virtual handler of the control "DialogDragEnd" event.

```
virtual bool OnDialogDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragEnd" event occurs at the end of the drag of the control.

CAppDialog

CAppDialog is a class of Application Dialog complex control (with dependent controls).

Description

CAppDialog class is intended to combine the controls with different functions in the group inside the MQL5 program.

Declaration

```
class CAppDialog : public CDialog
```

Title

```
#include <Controls\Dialog.mqh>
```

Inheritance hierarchy

[CObject](#)

[CWnd](#)

[CWndContainer](#)

[CDialog](#)

CAppDialog

Class Methods by Groups

Create and destroy	
Create	Creates control
Destroy	Destroys control
Events processing	
OnEvent	Event handler of all chart events
Run	
Run	Runs control
Chart events processing	
ChartEvent	Event handler of all chart events
Settings	
Minimized	Sets a value indicating whether the control is minimized
Save/Load	
IniFileSave	Saves the control state to file
IniFileLoad	Loads the control state from file

IniFileName	Sets the file name for loading/saving the control state
IniFileExt	Sets the file extension for loading/saving the control state
Initialization	
CreateCommon	Common initialization method
CreateExpert	Initialization method for working in Expert Advisors
CreateIndicator	Initialization method for working in indicators
Dependent controls	
CreateButtonMinMax	Creates dependent controls (minimize/maximize buttons)
Dependent controls event handlers	
OnClickButtonClose	"ClickButtonClose" internal event handler (virtual)
OnClickButtonMinMax	"ClickButtonMinMax" internal event handler (virtual)
External events	
OnAnotherApplicationClose	Event handler of external events (virtual)
Methods	
Rebound	Sets new coordinates of the control using CRect class coordinates
Minimize	Shows the control in the minimized state
Maximize	Shows the control in the maximized (restored) state
CreateInstanceld	Creates a unique Id for the names of the control objects
ProgramName	Gets the name of MQL5 program, at which the control is used
SubwinOff	Get the Y offset of the control subwindow

Methods inherited from class CObject

Prev, Prev, Next, Next, [Type](#), [Compare](#)

Methods inherited from class CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), Size, Size, Size, [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#),

[StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

Methods inherited from class CWndContainer

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Methods inherited from class CDialog

[Caption](#), [Caption](#), [Add](#), [Add](#)

Create

Creates new CAppDialog control.

```
virtual bool Create(  
    const long    chart,      // chart ID  
    const string  name,      // name  
    const int     subwin,    // chart subwindow  
    const int     x1,        // coordinate  
    const int     y1,        // coordinate  
    const int     x2,        // coordinate  
    const int     y2        // coordinate  
)
```

Parameters

chart

[in] ID of the chart, at which the ontrol is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the ontrol is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Destroy

Destroys control.

```
virtual bool Destroy()
```

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool OnEvent(  
    const int      id,           // ID  
    const long&    lparam,      // parameter  
    const double&  dparam,      // parameter  
    const string&  sparam       // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

Run

Runs control.

```
bool Run()
```

Return Value

true - successful, otherwise - false.

ChartEvent

Virtual handler of the control events.

```
virtual bool ChartEvent(  
    const int      id,           // ID  
    const long&    lparam,       // parameter  
    const double&  dparam,       // parameter  
    const string&  sparam        // parameter  
)
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of [long](#) type passed by reference.

dparam

[in] Event parameter of [double](#) type passed by reference.

sparam

[in] Event parameter of [string](#) type passed by reference.

Return Value

true - event processed, otherwise - false.

Minimized

Sets the value of "Minimized" (window state) property of the control.

```
bool Minimized(  
    const bool flag    // state  
)
```

Parameters

flag

[in] New state.

Return Value

true - successful, otherwise - false.

IniFileSave

Saves the control state to file.

```
void IniFileSave()
```

Return Value

true - successful, otherwise - false.

IniFileLoad

Loads the control state from file.

```
void IniFileLoad()
```

Return Value

true - successful, otherwise - false.

IniFileName

Sets the file name for loading/saving the control state.

```
virtual string IniFileName() const
```

Return Value

File name for loading/saving of the control state.

Note

The file name includes the name of the Expert Advisor/indicator and working symbol, on which MQL5 program is launched.

IniFileExt

Sets the file extension for loading/saving the control state.

```
virtual string IniFileExt() const
```

Return Value

File extension used for loading/saving of the control state.

CreateCommon

Common initialization method.

```
bool CreateCommon(  
    const long    chart,        // chart ID  
    const string  name,         // name  
    const int     subwin,       // chart subwindow  
)
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

Return Value

true - successful, otherwise - false.

CreateExpert

Initialization method for working in Expert Advisors.

```
bool CreateExpert (
    const int    x1,          // coordinate
    const int    y1,          // coordinate
    const int    x2,          // coordinate
    const int    y2          // coordinate
)
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

CreateIndicator

Initialization method for working in indicators.

```
bool CreateIndicator(  
    const int    x1,          // coordinate  
    const int    y1,          // coordinate  
    const int    x2,          // coordinate  
    const int    y2           // coordinate  
)
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

CreateButtonMinMax

Creates dependent controls (minimize/maximize buttons).

```
virtual void CreateButtonMinMax()
```

Return Value

None.

OnClickButtonClose

The virtual handler of the control "ClickButtonClose" (mouse click on close button) internal event.

```
virtual void OnClickButtonClose()
```

Return Value

None.

OnClickButtonMinMax

The virtual handler of the control "ClickButtonMinMax" (mouse click on minimize/maximize button) internal event.

```
virtual void OnClickButtonClose()
```

Return Value

None.

OnAnotherApplicationClose

The virtual handler of the control external events.

```
virtual void OnAnotherApplicationClose()
```

Return Value

None.

Rebound

Sets new coordinates of the control using CRect class coordinates.

```
bool Rebound(  
    const & CRect rect    // CRect class  
)
```

Return Value

true - successful, otherwise - false.

Minimize

Shows the control window in the minimized state.

```
virtual void Minimize()
```

Return Value

true - successful, otherwise - false.

Maximize

Shows the control window in the maximized (restored) state.

```
virtual void Maximize()
```

Return Value

true - successful, otherwise - false.

CreateInstanceId

Creates a unique prefix for the names of the control objects.

```
string CreateInstanceId()
```

Return Value

Prefix for the object names.

ProgramName

Gets the name of the MQL5 program, at which the control is used.

```
string ProgramName ()
```

Return Value

Name of the MQL5 program.

SubwinOff

Gets the Y offset of the control subwindow.

```
void SubwinOff()
```

Return Value

None.

Moving from MQL4 to MQL5

MQL5 is the evolution of its predecessor - the MQL4 programming language, in which numerous indicators, scripts, and Expert Advisors were written. Despite the fact that the new programming language is maximally compatible with the previous-generation language, there are still some differences between these languages. And when transferring programs these differences should be noted.

This section contains information intended to facilitate the adaptation of codes to the new MQL5 language for programmers who know MQL4.

First it should be noted:

- The new language does not contain functions `start()`, `init()` and `deinit()`.
- No limit for the number of indicator buffers.
- DLLs are loaded immediately after loading an Expert Advisor (or any other mql5 program).
- Check of logical conditions is shortened.
- When limits of an array are exceeded, the current performance is terminated (critically - with the output of an errors).
- Precedence of operators like in C + +.
- The language offers the implicit type cast (even from string to a number).
- Local variables are not initialized automatically (except for strings).
- Common local arrays are automatically deleted.

Special Functions `init`, `start` and `deinit`

The MQL4 language contained only three predefined functions that could be used in the indicator, script or Expert Advisor (not taking into account the include files `*.mqh` and library files). In MQL5 there are no such functions, but there are their analogues. The table shows the approximate correspondence of functions.

MQL4	MQL5
<code>init</code>	<code>OnInit</code>
<code>start</code>	<code>OnStart</code>
<code>deinit</code>	<code>OnDeinit</code>

Functions [OnInit](#) and [OnDeinit](#) perform the same role as `init` and `deinit` in MQL4 - they are designed to locate the code, which must be performed during initialization and deinitialization of mql5 programs. You can either just rename these functions accordingly, or leave them as they are, but add calls of these functions in corresponding places.

Example:

```
void OnInit()  
{  
    //--- Call function upon initialization  
    init();  
}
```



```
    }
    void OnDeinit(const int reason)
    {
        //--- Call function upon deinitialization
        deinit();
        //---
    }
```

The start function is replaced by [OnStart](#) only in scripts. In Expert Advisors and indicators it should be renamed to [OnTick](#) and [OnCalculate](#), respectively. The code that is to be executed during a mql5 program operation should be located in these three functions:

mql5-program	main function
script	OnStart
indicator	OnCalculate
Expert Advisor	OnTick

If the indicator or script code does not contain the main function, or the function name differs from the required one, the call of this function is not performed. It means, if the source code of a script doesn't contain OnStart, such a code will be compiled as an Expert Advisor.

If an indicator code doesn't contain the OnCalculate function, the compilation of such an indicator is impossible.

Predefined Variables

In MQL5 there are no such predefined variables as Ask, Bid, Bars. Variables Point and Digits have a slightly different spelling:

MQL4	MQL5
Digits	_Digits
Point	_Point
	_LastError
	_Period
	_Symbol
	_StopFlag
	_UninitReason

Access to Timeseries

In MQL5 there are no such predefined timeseries as Open[], High[], Low[], Close[], Volume[] and Time[]. The necessary depth of a timeseries can now be set using corresponding [functions to access timeseries](#).

Expert Advisors

Expert Advisors in MQL5 do not require the obligatory presence of functions that handle the [events](#) of a new tick receipt - OnTick, as it was in MQL4 (the start function in MQL4 is executed when a new tick is received). In MQL5 Expert Advisors can contain pre-defined handler functions of several types of events:

- [OnTick](#) - receipt of a new tick;
- [OnTimer](#) - timer event;
- [OnTrade](#) - trade event;
- [OnChartEvent](#) - events of input from the keyboard and mouse, events of a graphic object moving, event of a text editing completion in the entry field of the LabelEdit object;
- [OnBookEvent](#) - event of Depth of Market status change.

Custom Indicators

In MQL4, the number of indicator buffers is limited and can't exceed 8. In MQL5 there are no such limitations, but it should be remembered that each indicator buffer requires allocation of a certain part of memory for its location in the terminal, so the new possibility should not be abused.

MQL4 offered only 6 types of custom indicator plotting; while MQL5 now offers 18 [drawing styles](#). The names of drawing types haven't changed, but the ideology of the graphical representation of indicators has changed significantly.

The direction of indexing in indicator buffers also differs. By default, in MQL5 all the indicator buffers have the behavior of common arrays, i.e. 0 indexed element is the oldest one in the history, and as the index increases, we move from the oldest data to the most recent ones.

The only function for working with [custom indicators](#) that was preserved from MQL4 is [SetIndexBuffer](#). But its call has changed; now you should specify [type of data to be stored in an array](#), linked to the indicator buffer.

Properties of custom indicators also have changed and expanded. New functions for [accessing timeseries](#) have been added, so the total calculation algorithm must be reconsidered.

Graphical Objects

The number of graphical objects in MQL5 has been significantly increased. Besides, graphical objects can now be positioned in time with the accuracy of a second in a chart of any timeframe - now object anchor points are not rounded off to the bar opening time in the current price chart.

For the Arrow, Text and Label objects now you can specify [binding methods](#), and for the Label, Button, Chart, Bitmap Label and Edit objects you can set [chart corner to which an object is attached](#).

List of MQL5 Functions

All MQL5 functions in alphabetical order.

Function	Action	Section
<u>AccountInfoDouble</u>	Returns a value of double type of the corresponding account property	<u>Account Information</u>
<u>AccountInfoInteger</u>	Returns a value of integer type (bool, int or long) of the corresponding account property	<u>Account Information</u>
<u>AccountInfoString</u>	Returns a value string type corresponding account property	<u>Account Information</u>
<u>acos</u>	Returns the arc cosine of x in radians	<u>Math Functions</u>
<u>Alert</u>	Displays a message in a separate window	<u>Common Functions</u>
<u>ArrayBsearch</u>	Searches for a specified value in a multidimensional numeric array sorted ascending	<u>Array Functions</u>
<u>ArrayCompare</u>	Returns the result of comparing two arrays of <u>simple types</u> or custom structures without <u>complex objects</u>	<u>Array Functions</u>
<u>ArrayCopy</u>	Copies one array into another	<u>Array Functions</u>
<u>ArrayFill</u>	Fills an array with the specified value	<u>Array Functions</u>
<u>ArrayFree</u>	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.	<u>Array Functions</u>
<u>ArrayGetAsSeries</u>	Checks direction of array indexing	<u>Array Functions</u>
<u>ArrayInitialize</u>	Sets all elements of a numeric array into a single value	<u>Array Functions</u>
<u>ArrayIsDynamic</u>	Checks whether an array is dynamic	<u>Array Functions</u>
<u>ArrayIsSeries</u>	Checks whether an array is a timeseries	<u>Array Functions</u>

<u>ArrayMaximum</u>	Searches for the largest element in the first dimension of a multidimensional numeric array	<u>Array Functions</u>
<u>ArrayMinimum</u>	Searches for the lowest element in the first dimension of a multidimensional numeric array	<u>Array Functions</u>
<u>ArrayRange</u>	Returns the number of elements in the specified dimension of the array	<u>Array Functions</u>
<u>ArrayResize</u>	Sets the new size in the first dimension of the array	<u>Array Functions</u>
<u>ArraySetAsSeries</u>	Sets the direction of array indexing	<u>Array Functions</u>
<u>ArraySize</u>	Returns the number of elements in the array	<u>Array Functions</u>
<u>ArraySort</u>	Sorting of numeric arrays by the first dimension	<u>Array Functions</u>
<u>asin</u>	Returns the arc sine of x in radians	<u>Math Functions</u>
<u>atan</u>	Returns the arc tangent of x in radians	<u>Math Functions</u>
<u>Bars</u>	Returns the number of bars the history for a specified symbol and period	<u>Timeseries and Indicators Access</u>
<u>BarsCalculated</u>	Returns the number of calculated data in an indicator buffer or -1 in the case of error (data hasn't been calculated yet)	<u>Timeseries and Indicators Access</u>
<u>ceil</u>	Returns integer numeric value closest from above	<u>Math Functions</u>
<u>CharArrayToString</u>	Converting symbol code (ansi) into one-symbol array	<u>Conversion Functions</u>
<u>ChartApplyTemplate</u>	Applies a specific template from a specified file to the chart	<u>Chart Operations</u>
<u>ChartClose</u>	Closes the specified chart	<u>Chart Operations</u>
<u>ChartFirst</u>	Returns the ID of the first chart of the client terminal	<u>Chart Operations</u>

<u>ChartGetDouble</u>	Returns the double value property of the specified chart	<u>Chart Operations</u>
<u>ChartGetInteger</u>	Returns the integer value property of the specified chart	<u>Chart Operations</u>
<u>ChartGetString</u>	Returns the string value property of the specified chart	<u>Chart Operations</u>
<u>ChartID</u>	Returns the ID of the current chart	<u>Chart Operations</u>
<u>ChartIndicatorAdd</u>	Adds an indicator with the specified handle into a specified chart window	<u>Chart Operations</u>
<u>ChartIndicatorDelete</u>	Removes an indicator with a specified name from the specified chart window	<u>Chart Operations</u>
<u>ChartIndicatorGet</u>	Returns the handle of the indicator with the specified short name in the specified chart window	<u>Chart Operations</u>
<u>ChartIndicatorName</u>	Returns the short name of the indicator by the number in the indicators list on the specified chart window	<u>Chart Operations</u>
<u>ChartIndicatorsTotal</u>	Returns the number of all indicators applied to the specified chart window.	<u>Chart Operations</u>
<u>ChartNavigate</u>	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart	<u>Chart Operations</u>
<u>ChartNext</u>	Returns the chart ID of the chart next to the specified one	<u>Chart Operations</u>
<u>ChartOpen</u>	Opens a new chart with the specified symbol and period	<u>Chart Operations</u>
<u>CharToString</u>	Converting a symbol code into a one-character string	<u>Conversion Functions</u>
<u>ChartPeriod</u>	Returns the period value of the specified chart	<u>Chart Operations</u>
<u>ChartPriceOnDropped</u>	Returns the price coordinate of the chart point, the Expert Advisor or script has been dropped to	<u>Chart Operations</u>

<u>ChartRedraw</u>	Calls a forced redrawing of a specified chart	<u>Chart Operations</u>
<u>ChartSaveTemplate</u>	Saves current chart settings in a template with a specified name	<u>Chart Operations</u>
<u>ChartScreenShot</u>	Provides a screenshot of the chart of its current state in a GIF, PNG or BMP format depending on specified extension	<u>Chart Operations</u>
<u>ChartSetDouble</u>	Sets the double value for a corresponding property of the specified chart	<u>Chart Operations</u>
<u>ChartSetInteger</u>	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart	<u>Chart Operations</u>
<u>ChartSetString</u>	Sets the string value for a corresponding property of the specified chart	<u>Chart Operations</u>
<u>ChartSetSymbolPeriod</u>	Changes the symbol value and a period of the specified chart	<u>Chart Operations</u>
<u>ChartSymbol</u>	Returns the symbol name of the specified chart	<u>Chart Operations</u>
<u>ChartTimeOnDropped</u>	Returns the time coordinate of the chart point, the Expert Advisor or script has been dropped to	<u>Chart Operations</u>
<u>ChartTimePriceToXY</u>	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates	<u>Chart Operations</u>
<u>ChartWindowFind</u>	Returns the number of a subwindow where an indicator is drawn	<u>Chart Operations</u>
<u>ChartWindowOnDropped</u>	Returns the number (index) of the chart subwindow, the Expert Advisor or script has been dropped to	<u>Chart Operations</u>
<u>ChartXOnDropped</u>	Returns the X coordinate of the chart point, the Expert Advisor or script has been dropped to	<u>Chart Operations</u>

<u>ChartXYToTimePrice</u>	Converts the X and Y coordinates on a chart to the time and price values	<u>Chart Operations</u>
<u>ChartYOnDropped</u>	Returns the Y coordinate of the chart point, the Expert Advisor or script has been dropped to	<u>Chart Operations</u>
<u>CheckPointer</u>	Returns the type of the object pointer	<u>Common Functions</u>
<u>CLBufferCreate</u>	Creates an OpenCL buffer	<u>Working with OpenCL</u>
<u>CLBufferFree</u>	Deletes an OpenCL buffer	<u>Working with OpenCL</u>
<u>CLBufferRead</u>	Reads an OpenCL buffer into an array	<u>Working with OpenCL</u>
<u>CLBufferWrite</u>	Writes an array into an OpenCL buffer	<u>Working with OpenCL</u>
<u>CLContextCreate</u>	Creates an OpenCL context	<u>Working with OpenCL</u>
<u>CLContextFree</u>	Removes an OpenCL context	<u>Working with OpenCL</u>
<u>CLExecute</u>	Runs an OpenCL program	<u>Working with OpenCL</u>
<u>CLGetDeviceInfo</u>	Receives device property from OpenCL driver	<u>Working with OpenCL</u>
<u>CLGetInfoInteger</u>	Returns the value of an integer property for an OpenCL object or device	<u>Working with OpenCL</u>
<u>CLHandleType</u>	Returns the type of an OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration	<u>Working with OpenCL</u>
<u>CLKernelCreate</u>	Creates an OpenCL start function	<u>Working with OpenCL</u>
<u>CLKernelFree</u>	Removes an OpenCL start function	<u>Working with OpenCL</u>
<u>CLProgramCreate</u>	Creates an OpenCL program from a source code	<u>Working with OpenCL</u>
<u>CLProgramFree</u>	Removes an OpenCL program	<u>Working with OpenCL</u>
<u>CLSetKernelArg</u>	Sets a parameter for the OpenCL function	<u>Working with OpenCL</u>
<u>CLSetKernelArgMem</u>	Sets an OpenCL buffer as a parameter of the OpenCL function	<u>Working with OpenCL</u>

<u>ColorToARGB</u>	Converting color type to uint type to receive ARGB representation of the color.	<u>Conversion Functions</u>
<u>ColorToString</u>	Converting color value into string as "R,G,B"	<u>Conversion Functions</u>
<u>Comment</u>	Outputs a comment in the left top corner of the chart	<u>Common Functions</u>
<u>CopyBuffer</u>	Gets data of a specified buffer from a specified indicator into an array	<u>Timeseries and Indicators Access</u>
<u>CopyClose</u>	Gets history data on bar closing price for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyHigh</u>	Gets history data on maximal bar price for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyLow</u>	Gets history data on minimal bar price for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyOpen</u>	Gets history data on bar opening price for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyRates</u>	Gets history data of the <u>Rates</u> structure for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyRealVolume</u>	Gets history data on trade volumes for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopySpread</u>	Gets history data on spreads for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>CopyTicks</u>	Gets ticks accumulated by the terminal for the current working session into an array	<u>Timeseries and Indicators Access</u>
<u>CopyTickVolume</u>	Gets history data on tick volumes for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>

<u>CopyTime</u>	Gets history data on bar opening time for a specified symbol and period into an array	<u>Timeseries and Indicators Access</u>
<u>cos</u>	Returns the cosine of a number	<u>Math Functions</u>
<u>CryptDecode</u>	Performs the inverse transformation of the data from array	<u>Common Functions</u>
<u>CryptEncode</u>	Transforms the data from array with the specified method	<u>Common Functions</u>
<u>DebugBreak</u>	Program breakpoint in debugging	<u>Common Functions</u>
<u>Digits</u>	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol	<u>Checkup</u>
<u>DoubleToString</u>	Converting a numeric value to a text line with a specified accuracy	<u>Conversion Functions</u>
<u>EnumToString</u>	Converting an enumeration value of any type to string	<u>Conversion Functions</u>
<u>EventChartCustom</u>	Generates a custom event for the specified chart	<u>Working with Events</u>
<u>EventKillTimer</u>	Stops the generation of events by the timer in the current chart	<u>Working with Events</u>
<u>EventSetMillisecondTimer</u>	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart	<u>Working with Events</u>
<u>EventSetTimer</u>	Starts the timer event generator with the specified periodicity for the current chart	<u>Working with Events</u>
<u>exp</u>	Returns exponent of a number	<u>Math Functions</u>
<u>ExpertRemove</u>	Stops Expert Advisor and unloads it from the chart	<u>Common Functions</u>
<u>fabs</u>	Returns absolute value (modulus) of the specified numeric value	<u>Math Functions</u>

<u>FileClose</u>	Closes a previously opened file	<u>File Functions</u>
<u>FileCopy</u>	Copies the original file from a local or shared folder to another file	<u>File Functions</u>
<u>FileDelete</u>	Deletes a specified file	<u>File Functions</u>
<u>FileFindClose</u>	Closes search handle	<u>File Functions</u>
<u>FileFindFirst</u>	Starts the search of files in a directory in accordance with the specified filter	<u>File Functions</u>
<u>FileFindNext</u>	Continues the search started by the FileFindFirst() function	<u>File Functions</u>
<u>FileFlush</u>	Writes to a disk all data remaining in the input/output file buffer	<u>File Functions</u>
<u>FileGetInteger</u>	Gets an integer property of a file	<u>File Functions</u>
<u>FilesEnding</u>	Defines the end of a file in the process of reading	<u>File Functions</u>
<u>FilesExist</u>	Checks the existence of a file	<u>File Functions</u>
<u>FilesLineEnding</u>	Defines the end of a line in a text file in the process of reading	<u>File Functions</u>
<u>FileMove</u>	Moves or renames a file	<u>File Functions</u>
<u>FileOpen</u>	Opens a file with a specified name and flag	<u>File Functions</u>
<u>FileReadArray</u>	Reads arrays of any type except for string from the file of the BIN type	<u>File Functions</u>
<u>FileReadBool</u>	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type	<u>File Functions</u>
<u>FileReadDatetime</u>	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value	<u>File Functions</u>

<u>FileReadDouble</u>	Reads a double value from the current position of the file pointer	<u>File Functions</u>
<u>FileReadFloat</u>	Reads a float value from the current position of the file pointer	<u>File Functions</u>
<u>FileReadInteger</u>	Reads int, short or char value from the current position of the file pointer	<u>File Functions</u>
<u>FileReadLong</u>	Reads a long type value from the current position of the file pointer	<u>File Functions</u>
<u>FileReadNumber</u>	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string into double value	<u>File Functions</u>
<u>FileReadString</u>	Reads a string from the current position of a file pointer from a file	<u>File Functions</u>
<u>FileReadStruct</u>	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer	<u>File Functions</u>
<u>FileSeek</u>	Moves the position of the file pointer by a specified number of bytes relative to the specified position	<u>File Functions</u>
<u>FileSize</u>	Returns the size of a corresponding open file	<u>File Functions</u>
<u>FileTell</u>	Returns the current position of the file pointer of a corresponding open file	<u>File Functions</u>
<u>FileWrite</u>	Writes data to a file of CSV or TXT type	<u>File Functions</u>
<u>FileWriteArray</u>	Writes arrays of any type except for string into a file of BIN type	<u>File Functions</u>
<u>FileWriteDouble</u>	Writes value of the double type from the current position	<u>File Functions</u>

	of a file pointer into a binary file	
FileWriteFloat	Writes value of the float type from the current position of a file pointer into a binary file	File Functions
FileWriteInteger	Writes value of the int type from the current position of a file pointer into a binary file	File Functions
FileWriteLong	Writes value of the long type from the current position of a file pointer into a binary file	File Functions
FileWriteString	Writes the value of a string parameter into a BIN or TXT file starting from the current position of the file pointer	File Functions
FileWriteStruct	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer	File Functions
floor	Returns integer numeric value closest from below	Math Functions
fmax	Returns the maximal value of the two numeric values	Math Functions
fmin	Returns the minimal value of the two numeric values	Math Functions
fmod	Returns the real remainder after the division of two numbers	Math Functions
FolderClean	Deletes all files in the specified folder	File Functions
FolderCreate	Creates a folder in the Files directory	File Functions
FolderDelete	Removes a selected directory. If the folder is not empty, then it can't be removed	File Functions
FrameAdd	Adds a frame with data	Working with Optimization Results
FrameFilter	Sets the frame reading filter and moves the pointer to the beginning	Working with Optimization Results

<u>FrameFirst</u>	Moves a pointer of frame reading to the beginning and resets the previously set filter	<u>Working with Optimization Results</u>
<u>FrameInputs</u>	Receives <u>input parameters</u> , on which the frame is formed	<u>Working with Optimization Results</u>
<u>FrameNext</u>	Reads a frame and moves the pointer to the next one	<u>Working with Optimization Results</u>
<u>GetLastError</u>	Returns the last error	<u>Checkup</u>
<u>GetPointer</u>	Returns the object <u>pointer</u>	<u>Common Functions</u>
<u>GetTickCount</u>	Returns the number of milliseconds that have elapsed since the system was started	<u>Common Functions</u>
<u>GlobalVariableCheck</u>	Checks the existence of a global variable with the specified name	<u>Global Variables of the Terminal</u>
<u>GlobalVariableDel</u>	Deletes a global variable	<u>Global Variables of the Terminal</u>
<u>GlobalVariableGet</u>	Returns the value of a global variable	<u>Global Variables of the Terminal</u>
<u>GlobalVariableName</u>	Returns the name of a global variable by its ordinal number in the list of global variables	<u>Global Variables of the Terminal</u>
<u>GlobalVariablesDeleteAll</u>	Deletes global variables with the specified prefix in their names	<u>Global Variables of the Terminal</u>
<u>GlobalVariableSet</u>	Sets the new value to a global variable	<u>Global Variables of the Terminal</u>
<u>GlobalVariableSetOnCondition</u>	Sets the new value of the existing global variable by condition	<u>Global Variables of the Terminal</u>
<u>GlobalVariablesFlush</u>	Forcibly saves contents of all global variables to a disk	<u>Global Variables of the Terminal</u>
<u>GlobalVariablesTotal</u>	Returns the total number of global variables	<u>Global Variables of the Terminal</u>
<u>GlobalVariableTemp</u>	Sets the new value to a global variable, that exists only in the current session of the terminal	<u>Global Variables of the Terminal</u>
<u>GlobalVariableTime</u>	Returns time of the last accessing the global variable	<u>Global Variables of the Terminal</u>

<u>HistoryDealGetDouble</u>	Returns the requested property of a deal in the history (double)	<u>Trade Functions</u>
<u>HistoryDealGetInteger</u>	Returns the requested property of a deal in the history (datetime or int)	<u>Trade Functions</u>
<u>HistoryDealGetString</u>	Returns the requested property of a deal in the history (string)	<u>Trade Functions</u>
<u>HistoryDealGetTicket</u>	Returns a ticket of a corresponding deal in the history	<u>Trade Functions</u>
<u>HistoryDealSelect</u>	Selects a deal in the history for further calling it through appropriate functions	<u>Trade Functions</u>
<u>HistoryDealsTotal</u>	Returns the number of deals in the history	<u>Trade Functions</u>
<u>HistoryOrderGetDouble</u>	Returns the requested property of an order in the history (double)	<u>Trade Functions</u>
<u>HistoryOrderGetInteger</u>	Returns the requested property of an order in the history (datetime or int)	<u>Trade Functions</u>
<u>HistoryOrderGetString</u>	Returns the requested property of an order in the history (string)	<u>Trade Functions</u>
<u>HistoryOrderGetTicket</u>	Return order ticket of a corresponding order in the history	<u>Trade Functions</u>
<u>HistoryOrderSelect</u>	Selects an order in the history for further working with it	<u>Trade Functions</u>
<u>HistoryOrdersTotal</u>	Returns the number of orders in the history	<u>Trade Functions</u>
<u>HistorySelect</u>	Retrieves the history of transactions and orders for the specified period of the server time	<u>Trade Functions</u>
<u>HistorySelectByPosition</u>	Requests the history of deals with a specified <u>position identifier</u> .	<u>Trade Functions</u>
<u>iAC</u>	Accelerator Oscillator	<u>Technical Indicators</u>

<u>iAD</u>	Accumulation/Distribution	<u>Technical Indicators</u>
<u>iADX</u>	Average Directional Index	<u>Technical Indicators</u>
<u>iADXWilder</u>	Average Directional Index by Welles Wilder	<u>Technical Indicators</u>
<u>iAlligator</u>	Alligator	<u>Technical Indicators</u>
<u>iAMA</u>	Adaptive Moving Average	<u>Technical Indicators</u>
<u>iAO</u>	Awesome Oscillator	<u>Technical Indicators</u>
<u>iATR</u>	Average True Range	<u>Technical Indicators</u>
<u>iBands</u>	Bollinger Bands®	<u>Technical Indicators</u>
<u>iBearsPower</u>	Bears Power	<u>Technical Indicators</u>
<u>iBullsPower</u>	Bulls Power	<u>Technical Indicators</u>
<u>iBWMFI</u>	Market Facilitation Index by Bill Williams	<u>Technical Indicators</u>
<u>iCCI</u>	Commodity Channel Index	<u>Technical Indicators</u>
<u>iChaikin</u>	Chaikin Oscillator	<u>Technical Indicators</u>
<u>iCustom</u>	Custom indicator	<u>Technical Indicators</u>
<u>iDEMA</u>	Double Exponential Moving Average	<u>Technical Indicators</u>
<u>iDeMarker</u>	DeMarker	<u>Technical Indicators</u>
<u>iEnvelopes</u>	Envelopes	<u>Technical Indicators</u>
<u>iForce</u>	Force Index	<u>Technical Indicators</u>
<u>iFractals</u>	Fractals	<u>Technical Indicators</u>
<u>iFrAMA</u>	Fractal Adaptive Moving Average	<u>Technical Indicators</u>
<u>iGator</u>	Gator Oscillator	<u>Technical Indicators</u>
<u>ilchimoku</u>	Ichimoku Kinko Hyo	<u>Technical Indicators</u>
<u>iMA</u>	Moving Average	<u>Technical Indicators</u>
<u>iMACD</u>	Moving Averages Convergence-Divergence	<u>Technical Indicators</u>
<u>iMFI</u>	Money Flow Index	<u>Technical Indicators</u>
<u>iMomentum</u>	Momentum	<u>Technical Indicators</u>
<u>IndicatorCreate</u>	Returns the handle to the specified technical indicator	<u>Timeseries and Indicators Access</u>

	created by an array of MqlParam type parameters	
IndicatorParameters	Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters	Timeseries and Indicators Access
IndicatorRelease	Removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else	Timeseries and Indicators Access
IndicatorSetDouble	Sets the value of an indicator property of the double type	Custom Indicators
IndicatorSetInteger	Sets the value of an indicator property of the int type	Custom Indicators
IndicatorSetString	Sets the value of an indicator property of the string type	Custom Indicators
IntegerToString	Converting int into a string of preset length	Conversion Functions
iOBV	On Balance Volume	Technical Indicators
iOsMA	Moving Average of Oscillator (MACD histogram)	Technical Indicators
iRSI	Relative Strength Index	Technical Indicators
iRVI	Relative Vigor Index	Technical Indicators
iSAR	Parabolic Stop And Reverse System	Technical Indicators
IsStopped	Returns true, if an mql5 program has been commanded to stop its operation	Checkup
iStdDev	Standard Deviation	Technical Indicators
iStochastic	Stochastic Oscillator	Technical Indicators
iTEMA	Triple Exponential Moving Average	Technical Indicators
iTriX	Triple Exponential Moving Averages Oscillator	Technical Indicators
iVIDyA	Variable Index Dynamic Average	Technical Indicators
iVolumes	Volumes	Technical Indicators
iWPR	Williams' Percent Range	Technical Indicators

<u>log</u>	Returns natural logarithm	<u>Math Functions</u>
<u>log10</u>	Returns the logarithm of a number by base 10	<u>Math Functions</u>
<u>MarketBookAdd</u>	Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes	<u>Market Info</u>
<u>MarketBookGet</u>	Returns a structure array <u>MqlBookInfo</u> containing records of the Depth of Market of a specified symbol	<u>Market Info</u>
<u>MarketBookRelease</u>	Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes	<u>Market Info</u>
<u>MathAbs</u>	Returns absolute value (modulus) of the specified numeric value	<u>Math Functions</u>
<u>MathArccos</u>	Returns the arc cosine of x in radians	<u>Math Functions</u>
<u>MathArcsin</u>	Returns the arc sine of x in radians	<u>Math Functions</u>
<u>MathArctan</u>	Returns the arc tangent of x in radians	<u>Math Functions</u>
<u>MathCeil</u>	Returns integer numeric value closest from above	<u>Math Functions</u>
<u>MathCos</u>	Returns the cosine of a number	<u>Math Functions</u>
<u>MathExp</u>	Returns exponent of a number	<u>Math Functions</u>
<u>MathFloor</u>	Returns integer numeric value closest from below	<u>Math Functions</u>
<u>MathIsValidNumber</u>	Checks the correctness of a real number	<u>Math Functions</u>
<u>MathLog</u>	Returns natural logarithm	<u>Math Functions</u>
<u>MathLog10</u>	Returns the logarithm of a number by base 10	<u>Math Functions</u>
<u>MathMax</u>	Returns the maximal value of the two numeric values	<u>Math Functions</u>

<u>MathMin</u>	Returns the minimal value of the two numeric values	<u>Math Functions</u>
<u>MathMod</u>	Returns the real remainder after the division of two numbers	<u>Math Functions</u>
<u>MathPow</u>	Raises the base to the specified power	<u>Math Functions</u>
<u>MathRand</u>	Returns a pseudorandom value within the range of 0 to 32767	<u>Math Functions</u>
<u>MathRound</u>	Rounds of a value to the nearest integer	<u>Math Functions</u>
<u>MathSin</u>	Returns the sine of a number	<u>Math Functions</u>
<u>MathSqrt</u>	Returns a square root	<u>Math Functions</u>
<u>MathSrand</u>	Sets the starting point for generating a series of pseudorandom integers	<u>Math Functions</u>
<u>MathTan</u>	Returns the tangent of a number	<u>Math Functions</u>
<u>MessageBox</u>	Creates, displays a message box and manages it	<u>Common Functions</u>
<u>MQLInfoInteger</u>	Returns an integer value of a corresponding property of a running mql5 program	<u>Checkup</u>
<u>MQLInfoString</u>	Returns a string value of a corresponding property of a running mql5 program	<u>Checkup</u>
<u>NormalizeDouble</u>	Rounding of a floating point number to a specified accuracy	<u>Conversion Functions</u>
<u>ObjectCreate</u>	Creates an object of the specified type in a specified chart	<u>Object Functions</u>
<u>ObjectDelete</u>	Removes the object with the specified name from the specified chart (from the specified chart subwindow)	<u>Object Functions</u>
<u>ObjectFind</u>	Searches for an object with the specified ID by the name	<u>Object Functions</u>
<u>ObjectGetDouble</u>	Returns the double value of the corresponding object property	<u>Object Functions</u>

<u>ObjectGetInteger</u>	Returns the integer value of the corresponding object property	<u>Object Functions</u>
<u>ObjectGetString</u>	Returns the string value of the corresponding object property	<u>Object Functions</u>
<u>ObjectGetTimeByValue</u>	Returns the time value for the specified object price value	<u>Object Functions</u>
<u>ObjectGetValueByTime</u>	Returns the price value of an object for the specified time	<u>Object Functions</u>
<u>ObjectMove</u>	Changes the coordinates of the specified object anchor point	<u>Object Functions</u>
<u>ObjectName</u>	Returns the name of an object of the corresponding type in the specified chart (specified chart subwindow)	<u>Object Functions</u>
<u>ObjectsDeleteAll</u>	Removes all objects of the specified type from the specified chart (from the specified chart subwindow)	<u>Object Functions</u>
<u>ObjectSetDouble</u>	Sets the value of the corresponding object property	<u>Object Functions</u>
<u>ObjectSetInteger</u>	Sets the value of the corresponding object property	<u>Object Functions</u>
<u>ObjectSetString</u>	Sets the value of the corresponding object property	<u>Object Functions</u>
<u>ObjectsTotal</u>	Returns the number of objects of the specified type in the specified chart (specified chart subwindow)	<u>Object Functions</u>
<u>OrderCalcMargin</u>	Calculates the margin required for the specified order type, in the deposit currency	<u>Trade Functions</u>
<u>OrderCalcProfit</u>	Calculates the profit based on the parameters passed, in the deposit currency	<u>Trade Functions</u>
<u>OrderCheck</u>	Checks if there are enough funds to execute the required <u>trade operation</u> .	<u>Trade Functions</u>
<u>OrderGetDouble</u>	Returns the requested property of the order (double)	<u>Trade Functions</u>

OrderGetInteger	Returns the requested property of the order (datetime or int)	Trade Functions
OrderGetString	Returns the requested property of the order (string)	Trade Functions
OrderGetTicket	Return the ticket of a corresponding order	Trade Functions
OrderSelect	Selects a order for further working with it	Trade Functions
OrderSend	Sends trade requests to a server	Trade Functions
OrderSendAsync	Asynchronously sends trade requests without waiting for the trade response of the trade server	Trade Functions
OrdersTotal	Returns the number of orders	Trade Functions
ParameterGetRange	Receives data on the values range and the change step for an input variable when optimizing an Expert Advisor in the Strategy Tester	Working with Optimization Results
ParameterSetRange	Specifies the use of input variable when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values	Working with Optimization Results
Period	Returns the current chart timeframe	Checkup
PeriodSeconds	Returns the number of seconds in the period	Common Functions
PlaySound	Plays a sound file	Common Functions
PlotIndexGetInteger	Returns the value of an indicator line property of the integer type	Custom Indicators
PlotIndexSetDouble	Sets the value of an indicator line property of the type double	Custom Indicators
PlotIndexSetInteger	Sets the value of an indicator line property of the int type	Custom Indicators
PlotIndexSetString	Sets the value of an indicator line property of the string type	Custom Indicators

<u>Point</u>	Returns the point size of the current symbol in the quote currency	<u>Checkup</u>
<u>PositionGetDouble</u>	Returns the requested property of an open position (double)	<u>Trade Functions</u>
<u>PositionGetInteger</u>	Returns the requested property of an open position (datetime or int)	<u>Trade Functions</u>
<u>PositionGetString</u>	Returns the requested property of an open position (string)	<u>Trade Functions</u>
<u>PositionGetSymbol</u>	Returns the symbol corresponding to the open position	<u>Trade Functions</u>
<u>PositionGetTicket</u>	Returns the ticket of the position with the specified index in the list of open positions	<u>Trade Functions</u>
<u>PositionSelect</u>	Chooses an open position for further working with it	<u>Trade Functions</u>
<u>PositionSelectByTicket</u>	Selects a position to work with by the ticket number specified in it	<u>Trade Functions</u>
<u>PositionsTotal</u>	Returns the number of open positions	<u>Trade Functions</u>
<u>pow</u>	Raises the base to the specified power	<u>Math Functions</u>
<u>Print</u>	Displays a message in the log	<u>Common Functions</u>
<u>PrintFormat</u>	Formats and prints the sets of symbols and values in a log file in accordance with a preset format	<u>Common Functions</u>
<u>rand</u>	Returns a pseudorandom value within the range of 0 to 32767	<u>Math Functions</u>
<u>ResetLastError</u>	Sets the value of a predetermined variable <u>LastError</u> to zero	<u>Common Functions</u>
<u>ResourceCreate</u>	Creates an image resource based on a data set	<u>Common Functions</u>

<u>ResourceFree</u>	Deletes <u>dynamically created resource</u> (freeing the memory allocated for it)	<u>Common Functions</u>
<u>ResourceReadImage</u>	Reads data from the graphical resource <u>created by ResourceCreate() function</u> or <u>saved in EX5 file during compilation</u>	<u>Common Functions</u>
<u>ResourceSave</u>	Saves a resource into the specified file	<u>Common Functions</u>
<u>round</u>	Rounds of a value to the nearest integer	<u>Math Functions</u>
<u>SendFTP</u>	Sends a file at the address specified in the settings window of the "FTP" tab	<u>Common Functions</u>
<u>SendMail</u>	Sends an email at the address specified in the settings window of the "Email" tab	<u>Common Functions</u>
<u>SendNotification</u>	Sends push notifications to mobile terminals, whose MetaQuotes ID are specified in the "Notifications" tab	<u>Common Functions</u>
<u>SeriesInfoInteger</u>	Returns information about the state of historical data	<u>Timeseries and Indicators Access</u>
<u>SetIndexBuffer</u>	Binds the specified indicator buffer with one-dimensional dynamic <u>array</u> of the <u>double</u> type	<u>Custom Indicators</u>
<u>ShortArrayToString</u>	Copying array part into a string	<u>Conversion Functions</u>
<u>ShortToString</u>	Converting symbol code (unicode) into one-symbol string	<u>Conversion Functions</u>
<u>SignalBaseGetDouble</u>	Returns the value of double type property for selected signal	<u>Trade Signals</u>
<u>SignalBaseGetInteger</u>	Returns the value of integer type property for selected signal	<u>Trade Signals</u>
<u>SignalBaseGetString</u>	Returns the value of string type property for selected signal	<u>Trade Signals</u>

<u>SignalBaseSelect</u>	Selects a signal from signals, available in terminal for further working with it	<u>Trade Signals</u>
<u>SignalBaseTotal</u>	Returns the total amount of signals, available in terminal	<u>Trade Signals</u>
<u>SignalInfoGetDouble</u>	Returns the value of double type property of signal copy settings	<u>Trade Signals</u>
<u>SignalInfoGetInteger</u>	Returns the value of integer type property of signal copy settings	<u>Trade Signals</u>
<u>SignalInfoGetString</u>	Returns the value of string type property of signal copy settings	<u>Trade Signals</u>
<u>SignalInfoSetDouble</u>	Sets the value of double type property of signal copy settings	<u>Trade Signals</u>
<u>SignalInfoSetInteger</u>	Sets the value of integer type property of signal copy settings	<u>Trade Signals</u>
<u>SignalSubscribe</u>	Subscribes to the trading signal	<u>Trade Signals</u>
<u>SignalUnsubscribe</u>	Cancels subscription	<u>Trade Signals</u>
<u>sin</u>	Returns the sine of a number	<u>Math Functions</u>
<u>Sleep</u>	Suspends execution of the current Expert Advisor or script within a specified interval	<u>Common Functions</u>
<u>sqrt</u>	Returns a square root	<u>Math Functions</u>
<u>srand</u>	Sets the starting point for generating a series of pseudorandom integers	<u>Math Functions</u>
<u>StringAdd</u>	Adds a string to the end of another string	<u>String Functions</u>
<u>StringBufferLen</u>	Returns the size of buffer allocated for the string	<u>String Functions</u>
<u>StringCompare</u>	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1	<u>String Functions</u>

	(minus 1) - if the first string is less than the second one	
<u>StringConcatenate</u>	Forms a string of parameters passed	<u>String Functions</u>
<u>StringFill</u>	Fills out a specified string by selected symbols	<u>String Functions</u>
<u>StringFind</u>	Search for a substring in a string	<u>String Functions</u>
<u>StringFormat</u>	Converting number into string according to preset format	<u>Conversion Functions</u>
<u>StringGetCharacter</u>	Returns the value of a number located in the specified string position	<u>String Functions</u>
<u>StringInit</u>	Initializes string by specified symbols and provides the specified string length	<u>String Functions</u>
<u>StringLen</u>	Returns the number of symbols in a string	<u>String Functions</u>
<u>StringReplace</u>	Replaces all the found substrings of a string by a set sequence of symbols	<u>String Functions</u>
<u>StringSetCharacter</u>	Returns a copy of a string with a changed value of a symbol in a specified position	<u>String Functions</u>
<u>StringSplit</u>	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained	<u>String Functions</u>
<u>StringSubstr</u>	Extracts a substring from a text string starting from a specified position	<u>String Functions</u>
<u>StringToCharArray</u>	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type	<u>Conversion Functions</u>
<u>StringToColor</u>	Converting "R,G,B" string or string with color name into color type value	<u>Conversion Functions</u>
<u>StringToDouble</u>	Converting a string containing a symbol representation of number into number of double type	<u>Conversion Functions</u>

<u>StringToInteger</u>	Converting a string containing a symbol representation of number into number of int type	<u>Conversion Functions</u>
<u>StringToLower</u>	Transforms all symbols of a selected string to lowercase	<u>String Functions</u>
<u>StringToShortArray</u>	Symbol-wise copying a string to a selected part of array of ushort type	<u>Conversion Functions</u>
<u>StringToTime</u>	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type	<u>Conversion Functions</u>
<u>StringToUpper</u>	Transforms all symbols of a selected string into capitals	<u>String Functions</u>
<u>StringTrimLeft</u>	Cuts line feed characters, spaces and tabs in the left part of the string	<u>String Functions</u>
<u>StringTrimRight</u>	Cuts line feed characters, spaces and tabs in the right part of the string	<u>String Functions</u>
<u>StructToTime</u>	Converts a variable of MqlDateTime structure type into a datetime value	<u>Date and Time</u>
<u>Symbol</u>	Returns the name of a symbol of the current chart	<u>Checkup</u>
<u>SymbolInfoDouble</u>	Returns the double value of the symbol for the corresponding property	<u>Market Info</u>
<u>SymbolInfoInteger</u>	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property	<u>Market Info</u>
<u>SymbolInfoMarginRate</u>	Returns the margin rates depending on the order type and direction	<u>Market Info</u>
<u>SymbolInfoSessionQuote</u>	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.	<u>Market Info</u>

<u>SymbolInfoSessionTrade</u>	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.	<u>Market Info</u>
<u>SymbolInfoString</u>	Returns a value of the string type of a specified symbol for the corresponding property	<u>Market Info</u>
<u>SymbolInfoTick</u>	Returns the current prices for the specified symbol in a variable of the <u>MqlTick</u> type	<u>Market Info</u>
<u>SymbolsSynchronized</u>	Checks whether data of a selected symbol in the terminal are <u>synchronized</u> with data on the trade server	<u>Market Info</u>
<u>SymbolName</u>	Returns the name of a specified symbol	<u>Market Info</u>
<u>SymbolSelect</u>	Selects a symbol in the Market Watch window or removes a symbol from the window	<u>Market Info</u>
<u>SymbolsTotal</u>	Returns the number of available (selected in Market Watch or all) symbols	<u>Market Info</u>
<u>tan</u>	Returns the tangent of a number	<u>Math Functions</u>
<u>TerminalClose</u>	Commands the terminal to complete operation	<u>Common Functions</u>
<u>TerminalInfoDouble</u>	Returns an double value of a corresponding property of the mql5 program environment	<u>Checkup</u>
<u>TerminalInfoInteger</u>	Returns an integer value of a corresponding property of the mql5 program environment	<u>Checkup</u>
<u>TerminalInfoString</u>	Returns a string value of a corresponding property of the mql5 program environment	<u>Checkup</u>
<u>TesterStatistics</u>	It returns the value of a specified statistic calculated based on testing results	<u>Common Functions</u>
<u>TextGetSize</u>	Returns the string's width and height at the current <u>font settings</u>	<u>Object Functions</u>

<u>TextOut</u>	Transfers the text to the custom array (buffer) designed for creation of a graphical <u>resource</u>	<u>Object Functions</u>
<u>TextSetFont</u>	Sets the font for displaying the text using drawing methods (Arial 20 used by default)	<u>Object Functions</u>
<u>TimeCurrent</u>	Returns the last known server time (time of the last quote receipt) in the datetime format	<u>Date and Time</u>
<u>TimeDaylightSavings</u>	Returns the sign of Daylight Saving Time switch	<u>Date and Time</u>
<u>TimeGMT</u>	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running	<u>Date and Time</u>
<u>TimeGMTOffset</u>	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch	<u>Date and Time</u>
<u>TimeLocal</u>	Returns the local computer time in datetime format	<u>Date and Time</u>
<u>TimeToString</u>	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format	<u>Conversion Functions</u>
<u>TimeToStruct</u>	Converts a datetime value into a variable of MqlDateTime structure type	<u>Date and Time</u>
<u>TimeTradeServer</u>	Returns the current calculation time of the trade server	<u>Date and Time</u>
<u>UninitializeReason</u>	Returns the code of the reason for deinitialization	<u>Checkup</u>
<u>WebRequest</u>	Sends HTTP request to the specified server	<u>Common Functions</u>
<u>ZeroMemory</u>	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.	<u>Common Functions</u>

List of MQL5 Constants

All MQL5 constants in alphabetical order.

Constant	Description	Usage
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)	Print
__DATETIME__	File compilation date and time	Print
__FILE__	Name of the currently compiled file	Print
__FUNCSIG__	Signature of the function in whose body the macro is located. Logging of the full description of functions can be useful in the identification of overloaded functions	Print
__FUNCTION__	Name of the function, in whose body the macro is located	Print
__LINE__	Line number in the source code, in which the macro is located	Print
__MQLBUILD__, __MQL5BUILD__	Compiler build number	Print
__PATH__	An absolute path to the file that is currently being compiled	Print
ACCOUNT_ASSETS	The current assets of an account	AccountInfoDouble
ACCOUNT_BALANCE	Account balance in the deposit currency	AccountInfoDouble
ACCOUNT_COMMISSION_BLOCKED	The current blocked commission amount on an account	AccountInfoDouble
ACCOUNT_COMPANY	Name of a company that serves the account	AccountInfoString
ACCOUNT_CREDIT	Account credit in the deposit currency	AccountInfoDouble
ACCOUNT_CURRENCY	Account currency	AccountInfoString

ACCOUNT_EQUITY	Account equity in the deposit currency	AccountInfoDouble
ACCOUNT_LEVERAGE	Account leverage	AccountInfoInteger
ACCOUNT_LIABILITIES	The current liabilities on an account	AccountInfoDouble
ACCOUNT_LIMIT_ORDERS	Maximum allowed number of active pending orders	AccountInfoInteger
ACCOUNT_LOGIN	Account number	AccountInfoInteger
ACCOUNT_MARGIN	Account margin used in the deposit currency	AccountInfoDouble
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	AccountInfoDouble
ACCOUNT_MARGIN_INITIAL	Initial margin. The amount reserved on an account to cover the margin of all pending orders	AccountInfoDouble
ACCOUNT_MARGIN_LEVEL	Account margin level in percents	AccountInfoDouble
ACCOUNT_MARGIN_MAINTENANCE	Maintenance margin. The minimum equity reserved on an account to cover the minimum amount of all open positions	AccountInfoDouble
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	AccountInfoDouble
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	AccountInfoInteger
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	AccountInfoDouble
ACCOUNT_NAME	Client name	AccountInfoString
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	AccountInfoDouble
ACCOUNT_SERVER	Trade server name	AccountInfoString
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money	AccountInfoInteger

ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents	AccountInfoInteger
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	AccountInfoInteger
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert Advisor	AccountInfoInteger
ACCOUNT_TRADE_MODE	Account trade mode	AccountInfoInteger
ACCOUNT_TRADE_MODE_CONTEST	Contest account	AccountInfoInteger
ACCOUNT_TRADE_MODE_DEMO	Demo account	AccountInfoInteger
ACCOUNT_TRADE_MODE_REAL	Real account	AccountInfoInteger
ALIGN_CENTER	Centered (only for the Edit object)	ObjectSetInteger , ObjectGetInteger , ChartScreenShot
ALIGN_LEFT	Left alignment	ObjectSetInteger , ObjectGetInteger , ChartScreenShot
ALIGN_RIGHT	Right alignment	ObjectSetInteger , ObjectGetInteger , ChartScreenShot
ANCHOR_CENTER	Anchor point strictly in the center of the object	ObjectSetInteger , ObjectGetInteger
ANCHOR_LEFT	Anchor point to the left in the center	ObjectSetInteger , ObjectGetInteger
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner	ObjectSetInteger , ObjectGetInteger
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner	ObjectSetInteger , ObjectGetInteger
ANCHOR_LOWER	Anchor point below in the center	ObjectSetInteger , ObjectGetInteger
ANCHOR_RIGHT	Anchor point to the right in the center	ObjectSetInteger , ObjectGetInteger
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner	ObjectSetInteger , ObjectGetInteger
ANCHOR_RIGHT_UPPER	Anchor point at the upper right corner	ObjectSetInteger , ObjectGetInteger
ANCHOR_UPPER	Anchor point above in the center	ObjectSetInteger , ObjectGetInteger

BASE_LINE	Main line	Indicators Lines
BOOK_TYPE_BUY	Buy order (Bid)	MqlBookInfo
BOOK_TYPE_BUY_MARKET	Buy order by Market	MqlBookInfo
BOOK_TYPE_SELL	Sell order (Offer)	MqlBookInfo
BOOK_TYPE_SELL_MARKET	Sell order by Market	MqlBookInfo
BORDER_FLAT	Flat form	ObjectSetInteger , ObjectGetInteger
BORDER_RAISED	Prominent form	ObjectSetInteger , ObjectGetInteger
BORDER_SUNKEN	Concave form	ObjectSetInteger , ObjectGetInteger
CHAR_MAX	Maximal value, which can be represented by char type	Numerical Type Constants
CHAR_MIN	Minimal value, which can be represented by char type	Numerical Type Constants
CHART_AUTOSCROLL	Mode of automatic moving to the right border of the chart	ChartSetInteger , ChartGetInteger
CHART_BARS	Display as a sequence of bars	ChartSetInteger
CHART_BEGIN	Chart beginning (the oldest prices)	ChartNavigate
CHART_BRING_TO_TOP	Show chart on top of other charts	ChartSetInteger , ChartGetInteger
CHART_CANDLES	Display as Japanese candlesticks	ChartSetInteger
CHART_COLOR_ASK	Ask price level color	ChartSetInteger , ChartGetInteger
CHART_COLOR_BACKGROUND	Chart background color	ChartSetInteger , ChartGetInteger
CHART_COLOR_BID	Bid price level color	ChartSetInteger , ChartGetInteger
CHART_COLOR_CANDLE_BEAR	Body color of a bear candlestick	ChartSetInteger , ChartGetInteger
CHART_COLOR_CANDLE_BULL	Body color of a bull candlestick	ChartSetInteger , ChartGetInteger
CHART_COLOR_CHART_DOWN	Color for the down bar, shadows and body borders of bear candlesticks	ChartSetInteger , ChartGetInteger

<u>CHART_COLOR_CHART_LINE</u>	Line chart color and color of "Doji" Japanese candlesticks	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_CHART_UP</u>	Color for the up bar, shadows and body borders of bull candlesticks	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_FOREGROUND</u>	Color of axes, scales and OHLC line	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_GRID</u>	Grid color	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_LAST</u>	Line color of the last executed deal price (Last)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_STOP_LEVEL</u>	Color of stop order levels (Stop Loss and Take Profit)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COLOR_VOLUME</u>	Color of volumes and position opening levels	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_COMMENT</u>	Text of a comment in a chart	<u>ChartSetString</u> , <u>ChartGetString</u>
CHART_CURRENT_POS	Current position	<u>ChartNavigate</u>
<u>CHART_DRAG_TRADE_LEVELS</u>	Permission to drag trading levels on a chart with a mouse. The drag mode is enabled by default (true value)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_EVENT_MOUSE_MOVE</u>	Send notifications of mouse move and mouse click events (<u>CHARTEVENT_MOUSE_MOVE</u>) to all mql5 programs on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_EVENT_OBJECT_CREATE</u>	Send a notification of an event of new object creation (<u>CHARTEVENT_OBJECT_CREATE</u>) to all mql5-programs on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_EVENT_OBJECT_DELETE</u>	Send a notification of an event of object deletion (<u>CHARTEVENT_OBJECT_DELETE</u>) to all mql5-programs on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_FIRST_VISIBLE_BAR</u>	Number of the first visible bar in the chart. Indexing of bars is the same as for <u>timeseries</u> .	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_FIXED_MAX</u>	Fixed chart maximum	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>

<u>CHART_FIXED_MIN</u>	Fixed chart minimum	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>
<u>CHART_FIXED_POSITION</u>	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the horizontal time axis. It is displayed only if the automatic chart scrolling to the right on tick incoming is disabled (see <u>CHART_AUTOSCROLL</u> property). The bar on a fixed position remains in the same place when zooming in and out.	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>
<u>CHART_FOREGROUND</u>	Price chart in the foreground	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_HEIGHT_IN_PIXELS</u>	Chart height in pixels	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_IS_OBJECT</u>	Identifying "Chart" (<u>OBJ_CHART</u>) object - returns true for a graphical object. Returns false for a real chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_LINE</u>	Display as a line drawn by Close prices	<u>ChartSetInteger</u>
<u>CHART_MODE</u>	Chart type (candlesticks, bars or line)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_MOUSE_SCROLL</u>	Scrolling the chart horizontally using the left mouse button. Vertical scrolling is also available if the value of any following properties is set to true: <u>CHART_SCALEFIX</u> , <u>CHART_SCALEFIX_11</u> or <u>CHART_SCALE_PT_PER_BAR</u>	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_POINTS_PER_BAR</u>	Scale in points per bar	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>
<u>CHART_PRICE_MAX</u>	Chart maximum	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>
<u>CHART_PRICE_MIN</u>	Chart minimum	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>

<u>CHART_SCALE</u>	Scale	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SCALE_PT_PER_BAR</u>	Scale to be specified in points per bar	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SCALEFIX</u>	Fixed scale mode	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SCALEFIX_11</u>	Scale 1:1 mode	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHIFT</u>	Mode of price chart indent from the right border	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHIFT_SIZE</u>	The size of the zero bar indent from the right border in percents	<u>ChartSetDouble</u> , <u>ChartGetDouble</u>
<u>CHART_SHOW_ASK_LINE</u>	Display Ask values as a horizontal line in a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_BID_LINE</u>	Display Bid values as a horizontal line in a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_DATE_SCALE</u>	Showing the time scale on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_GRID</u>	Display grid in the chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_LAST_LINE</u>	Display Last values as a horizontal line in a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_OBJECT_DESCR</u>	Pop-up descriptions of graphical objects	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_OHLC</u>	Show OHLC values in the upper left corner	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_ONE_CLICK</u>	Showing the " <u>One click trading</u> " panel on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_PERIOD_SEP</u>	Display vertical separators between adjacent periods	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_PRICE_SCALE</u>	Showing the price scale on a chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_TRADE_LEVELS</u>	Displaying trade levels in the chart (levels of open positions, Stop Loss, Take Profit and pending orders)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_SHOW_VOLUMES</u>	Display volume in the chart	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>

<u>CHART_VISIBLE_BARS</u>	The number of bars on the chart that can be displayed	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
CHART_VOLUME_HIDE	Volumes are not shown	<u>ChartSetInteger</u>
CHART_VOLUME_REAL	Trade volumes	<u>ChartSetInteger</u>
CHART_VOLUME_TICK	Tick volumes	<u>ChartSetInteger</u>
<u>CHART_WIDTH_IN_BARS</u>	Chart width in bars	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_WIDTH_IN_PIXELS</u>	Chart width in pixels	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_WINDOW_HANDLE</u>	Chart window handle (HWND)	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_WINDOW_IS_VISIBLE</u>	Visibility of subwindows	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_WINDOW_YDISTANCE</u>	<p>The distance between the upper frame of the indicator subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a mouse event, the cursor coordinates are passed in terms of the coordinates of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow.</p> <p>The value is required for converting the absolute coordinates of the main chart to the local coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.</p>	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
<u>CHART_WINDOWS_TOTAL</u>	The total number of chart windows, including indicator subwindows	<u>ChartSetInteger</u> , <u>ChartGetInteger</u>
CHARTEVENT_CHART_CHANGE	Change of the chart size or modification of chart	<u>OnChartEvent</u>

	properties through the Properties dialog	
CHARTEVENT_CLICK	Clicking on a chart	OnChartEvent
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events	OnChartEvent
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events	OnChartEvent
CHARTEVENT_KEYDOWN	Keystrokes	OnChartEvent
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE =true is set for the chart)	OnChartEvent
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog	OnChartEvent
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object	OnChartEvent
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT_CREATE =true is set for the chart)	OnChartEvent
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE =true is set for the chart)	OnChartEvent
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object	OnChartEvent
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit	OnChartEvent
CHARTS_MAX	The maximum possible number of simultaneously open charts in the terminal	Other Constants
CHIKOSPAN_LINE	Chikou Span line	Indicators Lines
clrAliceBlue	Alice Blue	Web Colors
clrAntiqueWhite	Antique White	Web Colors
clrAqua	Aqua	Web Colors
clrAquamarine	Aquamarine	Web Colors
clrBeige	Beige	Web Colors
clrBisque	Bisque	Web Colors
clrBlack	Black	Web Colors
clrBlanchedAlmond	Blanched Almond	Web Colors

clrBlue	Blue	Web Colors
clrBlueViolet	Blue Violet	Web Colors
clrBrown	Brown	Web Colors
clrBurlyWood	Burly Wood	Web Colors
clrCadetBlue	Cadet Blue	Web Colors
clrChartreuse	Chartreuse	Web Colors
clrChocolate	Chocolate	Web Colors
clrCoral	Coral	Web Colors
clrCornflowerBlue	Cornflower Blue	Web Colors
clrCornsilk	Cornsilk	Web Colors
clrCrimson	Crimson	Web Colors
clrDarkBlue	Dark Blue	Web Colors
clrDarkGoldenrod	Dark Goldenrod	Web Colors
clrDarkGray	Dark Gray	Web Colors
clrDarkGreen	Dark Green	Web Colors
clrDarkKhaki	Dark Khaki	Web Colors
clrDarkOliveGreen	Dark Olive Green	Web Colors
clrDarkOrange	Dark Orange	Web Colors
clrDarkOrchid	Dark Orchid	Web Colors
clrDarkSalmon	Dark Salmon	Web Colors
clrDarkSeaGreen	Dark Sea Green	Web Colors
clrDarkSlateBlue	Dark Slate Blue	Web Colors
clrDarkSlateGray	Dark Slate Gray	Web Colors
clrDarkTurquoise	Dark Turquoise	Web Colors
clrDarkViolet	Dark Violet	Web Colors
clrDeepPink	Deep Pink	Web Colors
clrDeepSkyBlue	Deep Sky Blue	Web Colors
clrDimGray	Dim Gray	Web Colors
clrDodgerBlue	Dodger Blue	Web Colors
clrFireBrick	Fire Brick	Web Colors
clrForestGreen	Forest Green	Web Colors

clrGainsboro	Gainsboro	Web Colors
clrGold	Gold	Web Colors
clrGoldenrod	Goldenrod	Web Colors
clrGray	Gray	Web Colors
clrGreen	Green	Web Colors
clrGreenYellow	Green Yellow	Web Colors
clrHoneydew	Honeydew	Web Colors
clrHotPink	Hot Pink	Web Colors
clrIndianRed	Indian Red	Web Colors
clrIndigo	Indigo	Web Colors
clrIvory	Ivory	Web Colors
clrKhaki	Khaki	Web Colors
clrLavender	Lavender	Web Colors
clrLavenderBlush	Lavender Blush	Web Colors
clrLawnGreen	Lawn Green	Web Colors
clrLemonChiffon	Lemon Chiffon	Web Colors
clrLightBlue	Light Blue	Web Colors
clrLightCoral	Light Coral	Web Colors
clrLightCyan	Light Cyan	Web Colors
clrLightGoldenrod	Light Goldenrod	Web Colors
clrLightGray	Light Gray	Web Colors
clrLightGreen	Light Green	Web Colors
clrLightPink	Light Pink	Web Colors
clrLightSalmon	Light Salmon	Web Colors
clrLightSeaGreen	Light Sea Green	Web Colors
clrLightSkyBlue	Light Sky Blue	Web Colors
clrLightSlateGray	Light Slate Gray	Web Colors
clrLightSteelBlue	Light Steel Blue	Web Colors
clrLightYellow	Light Yellow	Web Colors
clrLime	Lime	Web Colors
clrLimeGreen	Lime Green	Web Colors

clrLinen	Linen	Web Colors
clrMagenta	Magenta	Web Colors
clrMaroon	Maroon	Web Colors
clrMediumAquaMarine	Medium Aquamarine	Web Colors
clrMediumBlue	Medium Blue	Web Colors
clrMediumOrchid	Medium Orchid	Web Colors
clrMediumPurple	Medium Purple	Web Colors
clrMediumSeaGreen	Medium Sea Green	Web Colors
clrMediumSlateBlue	Medium Slate Blue	Web Colors
clrMediumSpringGreen	Medium Spring Green	Web Colors
clrMediumTurquoise	Medium Turquoise	Web Colors
clrMediumVioletRed	Medium Violet Red	Web Colors
clrMidnightBlue	Midnight Blue	Web Colors
clrMintCream	Mint Cream	Web Colors
clrMistyRose	Misty Rose	Web Colors
clrMoccasin	Moccasin	Web Colors
clrNavajoWhite	Navajo White	Web Colors
clrNavy	Navy	Web Colors
clrNONE	Absence of color	Other Constants
clrOldLace	Old Lace	Web Colors
clrOlive	Olive	Web Colors
clrOliveDrab	Olive Drab	Web Colors
clrOrange	Orange	Web Colors
clrOrangeRed	Orange Red	Web Colors
clrOrchid	Orchid	Web Colors
clrPaleGoldenrod	Pale Goldenrod	Web Colors
clrPaleGreen	Pale Green	Web Colors
clrPaleTurquoise	Pale Turquoise	Web Colors
clrPaleVioletRed	Pale Violet Red	Web Colors
clrPapayaWhip	Papaya Whip	Web Colors
clrPeachPuff	Peach Puff	Web Colors

clrPeru	Peru	Web Colors
clrPink	Pink	Web Colors
clrPlum	Plum	Web Colors
clrPowderBlue	Powder Blue	Web Colors
clrPurple	Purple	Web Colors
clrRed	Red	Web Colors
clrRosyBrown	Rosy Brown	Web Colors
clrRoyalBlue	Royal Blue	Web Colors
clrSaddleBrown	Saddle Brown	Web Colors
clrSalmon	Salmon	Web Colors
clrSandyBrown	Sandy Brown	Web Colors
clrSeaGreen	Sea Green	Web Colors
clrSeashell	Seashell	Web Colors
clrSienna	Sienna	Web Colors
clrSilver	Silver	Web Colors
clrSkyBlue	Sky Blue	Web Colors
clrSlateBlue	Slate Blue	Web Colors
clrSlateGray	Slate Gray	Web Colors
clrSnow	Snow	Web Colors
clrSpringGreen	Spring Green	Web Colors
clrSteelBlue	Steel Blue	Web Colors
clrTan	Tan	Web Colors
clrTeal	Teal	Web Colors
clrThistle	Thistle	Web Colors
clrTomato	Tomato	Web Colors
clrTurquoise	Turquoise	Web Colors
clrViolet	Violet	Web Colors
clrWheat	Wheat	Web Colors
clrWhite	White	Web Colors
clrWhiteSmoke	White Smoke	Web Colors
clrYellow	Yellow	Web Colors

clrYellowGreen	Yellow Green	Web Colors
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart	ObjectSetInteger , ObjectGetInteger
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart	ObjectSetInteger , ObjectGetInteger
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart	ObjectSetInteger , ObjectGetInteger
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner of the chart	ObjectSetInteger , ObjectGetInteger
CP_ACP	The current Windows ANSI code page.	CharArrayToString , StringToCharArray , FileOpen
CP_MACCP	The current system Macintosh code page. Note: This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unicode for encoding.	CharArrayToString , StringToCharArray , FileOpen
CP_OEMCP	The current system OEM code page.	CharArrayToString , StringToCharArray , FileOpen
CP_SYMBOL	Symbol code page	CharArrayToString , StringToCharArray , FileOpen
CP_THREAD_ACP	The Windows ANSI code page for the current thread.	CharArrayToString , StringToCharArray , FileOpen
CP_UTF7	UTF-7 code page.	CharArrayToString , StringToCharArray , FileOpen
CP_UTF8	UTF-8 code page.	CharArrayToString , StringToCharArray , FileOpen
CRYPT_AES128	AES encryption with 128 bit key (16 bytes)	CryptEncode , CryptDecode
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)	CryptEncode , CryptDecode
CRYPT_ARCH_ZIP	ZIP archives	CryptEncode , CryptDecode
CRYPT_BASE64	BASE64	CryptEncode , CryptDecode
CRYPT_DES	DES encryption with 56 bit key (7 bytes)	CryptEncode , CryptDecode
CRYPT_HASH_MD5	MD5 HASH calculation	CryptEncode , CryptDecode
CRYPT_HASH_SHA1	SHA1 HASH calculation	CryptEncode , CryptDecode

CRYPT_HASH_SHA256	SHA256 HASH calculation	CryptEncode , CryptDecode
DBL_DIG	Number of significant decimal digits for double type	Numerical Type Constants
DBL_EPSILON	Minimal value, which satisfies the condition: $1.0 + \text{DBL_EPSILON} \neq 1.0$ (for double type)	Numerical Type Constants
DBL_MANT_DIG	Bits count in a mantissa for double type	Numerical Type Constants
DBL_MAX	Maximal value, which can be represented by double type	Numerical Type Constants
DBL_MAX_10_EXP	Maximal decimal value of exponent degree for double type	Numerical Type Constants
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	Numerical Type Constants
DBL_MIN	Minimal positive value, which can be represented by double type	Numerical Type Constants
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	Numerical Type Constants
DBL_MIN_EXP	Minimal binary value of exponent degree for double type	Numerical Type Constants
DEAL_COMMENT	Deal comment	HistoryDealGetString
DEAL_COMMISSION	Deal commission	HistoryDealGetDouble
DEAL_ENTRY	Deal entry - entry in, entry out, reverse	HistoryDealGetInteger
DEAL_ENTRY_IN	Entry in	HistoryDealGetInteger
DEAL_ENTRY_INOUT	Reverse	HistoryDealGetInteger
DEAL_ENTRY_OUT	Entry out	HistoryDealGetInteger
DEAL_MAGIC	Deal magic number (see ORDER_MAGIC)	HistoryDealGetInteger
DEAL_ORDER	Deal order number	HistoryDealGetInteger
DEAL_POSITION_ID	Identifier of a position , in the opening, modification or change of which this deal took	HistoryDealGetInteger

	part. Each position has a unique identifier that is assigned to all deals executed for the symbol during the entire lifetime of the position.	
DEAL_PRICE	Deal price	HistoryDealGetDouble
DEAL_PROFIT	Deal profit	HistoryDealGetDouble
DEAL_SWAP	Cumulative swap on close	HistoryDealGetDouble
DEAL_SYMBOL	Deal symbol	HistoryDealGetString
DEAL_TIME	Deal time	HistoryDealGetInteger
DEAL_TIME_MSC	The time of a deal execution in milliseconds since 01.01.1970	HistoryDealGetInteger
DEAL_TYPE	Deal type	HistoryDealGetInteger
DEAL_TYPE_BALANCE	Balance	HistoryDealGetInteger
DEAL_TYPE_BONUS	Bonus	HistoryDealGetInteger
DEAL_TYPE_BUY	Buy	HistoryDealGetInteger
DEAL_TYPE_BUY_CANCELED	Canceled buy deal. There can be a situation when a previously executed buy deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_BUY) is changed to DEAL_TYPE_BUY_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation	HistoryDealGetInteger
DEAL_TYPE_CHARGE	Additional charge	HistoryDealGetInteger
DEAL_TYPE_COMMISSION	Additional commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_AGENT_DAILY	Daily agent commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Monthly agent commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_DAILY	Daily commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_MONTHLY	Monthly commission	HistoryDealGetInteger

DEAL_TYPE_CORRECTION	Correction	HistoryDealGetInteger
DEAL_TYPE_CREDIT	Credit	HistoryDealGetInteger
DEAL_TYPE_INTEREST	Interest rate	HistoryDealGetInteger
DEAL_TYPE_SELL	Sell	HistoryDealGetInteger
DEAL_TYPE_SELL_CANCELED	Canceled sell deal. There can be a situation when a previously executed sell deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_SELL) is changed to DEAL_TYPE_SELL_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation	HistoryDealGetInteger
DEAL_VOLUME	Deal volume	HistoryDealGetDouble
DRAW_ARROW	Drawing arrows	Drawing Styles
DRAW_BARS	Display as a sequence of bars	Drawing Styles
DRAW_CANDLES	Display as a sequence of candlesticks	Drawing Styles
DRAW_COLOR_ARROW	Drawing multicolored arrows	Drawing Styles
DRAW_COLOR_BARS	Multicolored bars	Drawing Styles
DRAW_COLOR_CANDLES	Multicolored candlesticks	Drawing Styles
DRAW_COLOR_HISTOGRAM	Multicolored histogram from the zero line	Drawing Styles
DRAW_COLOR_HISTOGRAM2	Multicolored histogram of the two indicator buffers	Drawing Styles
DRAW_COLOR_LINE	Multicolored line	Drawing Styles
DRAW_COLOR_SECTION	Multicolored section	Drawing Styles
DRAW_COLOR_ZIGZAG	Multicolored ZigZag	Drawing Styles
DRAW_FILLING	Color fill between the two levels	Drawing Styles
DRAW_HISTOGRAM	Histogram from the zero line	Drawing Styles
DRAW_HISTOGRAM2	Histogram of the two indicator buffers	Drawing Styles
DRAW_LINE	Line	Drawing Styles

<u>DRAW_NONE</u>	Not drawn	<u>Drawing Styles</u>
<u>DRAW_SECTION</u>	Section	<u>Drawing Styles</u>
<u>DRAW_ZIGZAG</u>	Style Zigzag allows vertical section on the bar	<u>Drawing Styles</u>
ELLIOTT_CYCLE	Cycle	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_GRAND_SUPERCYCLE	Grand Supercycle	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_INTERMEDIATE	Intermediate	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_MINOR	Minor	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_MINUETTE	Minuette	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_MINUTE	Minute	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_PRIMARY	Primary	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_SUBMINUETTE	Subminuette	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
ELLIOTT_SUPERCYCLE	Supercycle	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
EMPTY_VALUE	Empty value in an indicator buffer	<u>Other Constants</u>
ERR_ACCOUNT_WRONG_PROPERTY	Wrong account property ID	<u>GetLastError</u>
ERR_ARRAY_BAD_SIZE	Requested array size exceeds 2 GB	<u>GetLastError</u>
ERR_ARRAY_RESIZE_ERROR	Not enough memory for the relocation of an array, or an attempt to change the size of a static array	<u>GetLastError</u>
ERR_BOOKS_CANNOT_ADD	Depth Of Market can not be added	<u>GetLastError</u>
ERR_BOOKS_CANNOT_DELETE	Depth Of Market can not be removed	<u>GetLastError</u>
ERR_BOOKS_CANNOT_GET	The data from Depth Of Market can not be obtained	<u>GetLastError</u>

ERR_BOOKS_CANNOT_SUBSCRIBE	Error in subscribing to receive new data from Depth Of Market	GetLastError
ERR_BUFFERS_NO_MEMORY	Not enough memory for the distribution of indicator buffers	GetLastError
ERR_BUFFERS_WRONG_INDEX	Wrong indicator buffer index	GetLastError
ERR_CANNOT_CLEAN_DIRECTORY	Failed to clear the directory (probably one or more files are blocked and removal operation failed)	GetLastError
ERR_CANNOT_DELETE_DIRECTORY	The directory cannot be removed	GetLastError
ERR_CANNOT_DELETE_FILE	File deleting error	GetLastError
ERR_CANNOT_OPEN_FILE	File opening error	GetLastError
ERR_CHAR_ARRAY_ONLY	Must be an array of type char	GetLastError
ERR_CHART_CANNOT_CHANGE	Failed to change chart symbol and period	GetLastError
ERR_CHART_CANNOT_CREATE_TIMER	Failed to create timer	GetLastError
ERR_CHART_CANNOT_OPEN	Chart opening error	GetLastError
ERR_CHART_INDICATOR_CANNOT_ADD	Error adding an indicator to chart	GetLastError
ERR_CHART_INDICATOR_CANNOT_DELETE	Error deleting an indicator from the chart	GetLastError
ERR_CHART_INDICATOR_NOT_FOUND	Indicator not found on the specified chart	GetLastError
ERR_CHART_NAVIGATE_FAILED	Error navigating through chart	GetLastError
ERR_CHART_NO_EXPERT	No Expert Advisor in the chart that could handle the event	GetLastError
ERR_CHART_NO_REPLY	Chart does not respond	GetLastError
ERR_CHART_NOT_FOUND	Chart not found	GetLastError
ERR_CHART_SCREENSHOT_FAILED	Error creating screenshots	GetLastError
ERR_CHART_TEMPLATE_FAILED	Error applying template	GetLastError

ERR_CHART_WINDOW_NOT_FOUND	Subwindow containing the indicator was not found	GetLastError
ERR_CHART_WRONG_ID	Wrong chart ID	GetLastError
ERR_CHART_WRONG_PARAMETER	Error value of the parameter for the function of working with charts	GetLastError
ERR_CHART_WRONG_PROPERTY	Wrong chart property ID	GetLastError
ERR_CUSTOM_WRONG_PROPERTY	Wrong ID of the custom indicator property	GetLastError
ERR_DIRECTORY_NOT_EXIST	Directory does not exist	GetLastError
ERR_DOUBLE_ARRAY_ONLY	Must be an array of type double	GetLastError
ERR_FILE_BINSTRINGSIZE	String size must be specified, because the file is opened as binary	GetLastError
ERR_FILE_CACHEBUFFER_ERROR	Not enough memory for cache to read	GetLastError
ERR_FILE_CANNOT_REWRITE	File can not be rewritten	GetLastError
ERR_FILE_IS_DIRECTORY	This is not a file, this is a directory	GetLastError
ERR_FILE_ISNOT_DIRECTORY	This is a file, not a directory	GetLastError
ERR_FILE_NOT_EXIST	File does not exist	GetLastError
ERR_FILE_NOTBIN	The file must be opened as a binary one	GetLastError
ERR_FILE_NOTCSV	The file must be opened as CSV	GetLastError
ERR_FILE_NOTTOREAD	The file must be opened for reading	GetLastError
ERR_FILE_NOTTOWRITE	The file must be opened for writing	GetLastError
ERR_FILE_NOTTXT	The file must be opened as a text	GetLastError
ERR_FILE_NOTTXTORCSV	The file must be opened as a text or CSV	GetLastError
ERR_FILE_READERROR	File reading error	GetLastError
ERR_FILE_WRITEERROR	Failed to write a resource to a file	GetLastError

ERR_FLOAT_ARRAY_ONLY	Must be an array of type float	GetLastError
ERR_FTP_SEND_FAILED	File sending via ftp failed	GetLastError
ERR_FUNCTION_NOT_ALLOWED	Function is not allowed for call	GetLastError
ERR_GLOBALVARIABLE_EXISTS	Global variable of the client terminal with the same name already exists	GetLastError
ERR_GLOBALVARIABLE_NOT_FOUND	Global variable of the client terminal is not found	GetLastError
ERR_HISTORY_NOT_FOUND	Requested history not found	GetLastError
ERR_HISTORY_WRONG_PROPERTY	Wrong ID of the history property	GetLastError
ERR_INCOMPATIBLE_ARRAYS	Copying incompatible arrays. String array can be copied only to a string array, and a numeric array - in numeric array only	GetLastError
ERR_INCOMPATIBLE_FILE	A text file must be for string arrays, for other arrays - binary	GetLastError
ERR_INDICATOR_CANNOT_ADD	Error applying an indicator to chart	GetLastError
ERR_INDICATOR_CANNOT_APPLY	The indicator cannot be applied to another indicator	GetLastError
ERR_INDICATOR_CANNOT_CREATE	Indicator cannot be created	GetLastError
ERR_INDICATOR_CUSTOM_NAME	The first parameter in the array must be the name of the custom indicator	GetLastError
ERR_INDICATOR_DATA_NOT_FOUND	Requested data not found	GetLastError
ERR_INDICATOR_NO_MEMORY	Not enough memory to add the indicator	GetLastError
ERR_INDICATOR_PARAMETER_TYPE	Invalid parameter type in the array when creating an indicator	GetLastError
ERR_INDICATOR_PARAMETERS_MISSING	No parameters when creating an indicator	GetLastError
ERR_INDICATOR_UNKNOWN_SYMBOL	Unknown symbol	GetLastError

ERR_INDICATOR_WRONG_HANDLE	Wrong indicator handle	GetLastError
ERR_INDICATOR_WRONG_INDEX	Wrong index of the requested indicator buffer	GetLastError
ERR_INDICATOR_WRONG_PARAMETERS	Wrong number of parameters when creating an indicator	GetLastError
ERR_INT_ARRAY_ONLY	Must be an array of type int	GetLastError
ERR_INTERNAL_ERROR	Unexpected internal error	GetLastError
ERR_INVALID_ARRAY	Array of a wrong type, wrong size, or a damaged object of a dynamic array	GetLastError
ERR_INVALID_DATETIME	Invalid date and/or time	GetLastError
ERR_INVALID_FILEHANDLE	A file with this handle was closed, or was not opening at all	GetLastError
ERR_INVALID_PARAMETER	Wrong parameter when calling the system function	GetLastError
ERR_INVALID_POINTER	Wrong pointer	GetLastError
ERR_INVALID_POINTER_TYPE	Wrong type of pointer	GetLastError
ERR_LONG_ARRAY_ONLY	Must be an array of type long	GetLastError
ERR_MAIL_SEND_FAILED	Email sending failed	GetLastError
ERR_MARKET_LASTTIME_UNKNOWN	Time of the last tick is not known (no ticks)	GetLastError
ERR_MARKET_NOT_SELECTED	Symbol is not selected in MarketWatch	GetLastError
ERR_MARKET_SELECT_ERROR	Error adding or deleting a symbol in MarketWatch	GetLastError
ERR_MARKET_UNKNOWN_SYMBOL	Unknown symbol	GetLastError
ERR_MARKET_WRONG_PROPERTY	Wrong identifier of a symbol property	GetLastError
ERR_MQL5_WRONG_PROPERTY	Wrong identifier of the program property	GetLastError
ERR_NO_STRING_DATE	No date in the string	GetLastError
ERR_NOT_ENOUGH_MEMORY	Not enough memory to perform the system function	GetLastError

ERR_NOTIFICATION_SEND_FAILED	Failed to send a notification	GetLastError
ERR_NOTIFICATION_TOO_FREQUENT	Too frequent sending of notifications	GetLastError
ERR_NOTIFICATION_WRONG_PARAMETER	Invalid parameter for sending a notification - an empty string or NULL has been passed to the SendNotification() function	GetLastError
ERR_NOTIFICATION_WRONG_SETTINGS	Wrong settings of notifications in the terminal (ID is not specified or permission is not set)	GetLastError
ERR_NOTINITIALIZED_STRING	Not initialized string	GetLastError
ERR_NUMBER_ARRAYS_ONLY	Must be a numeric array	GetLastError
ERR_OBJECT_ERROR	Error working with a graphical object	GetLastError
ERR_OBJECT_GETDATE_FAILED	Unable to get date corresponding to the value	GetLastError
ERR_OBJECT_GETVALUE_FAILED	Unable to get value corresponding to the date	GetLastError
ERR_OBJECT_NOT_FOUND	Graphical object was not found	GetLastError
ERR_OBJECT_WRONG_PROPERTY	Wrong ID of a graphical object property	GetLastError
ERR_ONEDIM_ARRAYS_ONLY	Must be a one-dimensional array	GetLastError
ERR_OPENCL_BUFFER_CREATE	Failed to create an OpenCL buffer	GetLastError
ERR_OPENCL_CONTEXT_CREATE	Error creating the OpenCL context	GetLastError
ERR_OPENCL_EXECUTE	OpenCL program runtime error	GetLastError
ERR_OPENCL_INTERNAL	Internal error occurred when running OpenCL	GetLastError
ERR_OPENCL_INVALID_HANDLE	Invalid OpenCL handle	GetLastError
ERR_OPENCL_KERNEL_CREATE	Error creating an OpenCL kernel	GetLastError
ERR_OPENCL_NOT_SUPPORTED	OpenCL functions are not supported on this computer	GetLastError

ERR_OPENCL_PROGRAM_CREATE	Error occurred when compiling an OpenCL program	GetLastError
ERR_OPENCL_QUEUE_CREATE	Failed to create a run queue in OpenCL	GetLastError
ERR_OPENCL_SET_KERNEL_PARAMETER	Error occurred when setting parameters for the OpenCL kernel	GetLastError
ERR_OPENCL_TOO_LONG_KERNEL_NAME	Too long kernel name (OpenCL kernel)	GetLastError
ERR_OPENCL_WRONG_BUFFER_OFFSET	Invalid offset in the OpenCL buffer	GetLastError
ERR_OPENCL_WRONG_BUFFER_SIZE	Invalid size of the OpenCL buffer	GetLastError
ERR_PLAY_SOUND_FAILED	Sound playing failed	GetLastError
ERR_RESOURCE_NAME_DUPLICATED	The names of the dynamic and the static resource match	GetLastError
ERR_RESOURCE_NAME_IS_TOO_LONG	The resource name exceeds 63 characters	GetLastError
ERR_RESOURCE_NOT_FOUND	Resource with this name has not been found in EX5	GetLastError
ERR_RESOURCE_UNSUPPORTED_TYPE	Unsupported resource type or its size exceeds 16 Mb	GetLastError
ERR_SERIES_ARRAY	Timeseries cannot be used	GetLastError
ERR_SHORT_ARRAY_ONLY	Must be an array of type short	GetLastError
ERR_SMALL_ARRAY	Too small array, the starting position is outside the array	GetLastError
ERR_SMALL_ASERIES_ARRAY	The receiving array is declared as AS_SERIES, and it is of insufficient size	GetLastError
ERR_STRING_OUT_OF_MEMORY	Not enough memory for the string	GetLastError
ERR_STRING_RESIZE_ERROR	Not enough memory for the relocation of string	GetLastError
ERR_STRING_SMALL_LEN	The string length is less than expected	GetLastError
ERR_STRING_TIME_ERROR	Error converting string to date	GetLastError
ERR_STRING_TOO_BIGNUMBER	Too large number, more than ULONG_MAX	GetLastError

ERR_STRING_UNKNOWNTYPE	Unknown data type when converting to a string	GetLastError
ERR_STRING_ZEROADDED	0 added to the string end, a useless operation	GetLastError
ERR_STRINGPOS_OUTOFRANGE	Position outside the string	GetLastError
ERR_STRUCT_WITHOBJECTS_ORCLASS	The structure contains objects of strings and/or dynamic arrays and/or structure of such objects and/or classes	GetLastError
ERR_SUCCESS	The operation completed successfully	GetLastError
ERR_TERMINAL_WRONG_PROPERTY	Wrong identifier of the terminal property	GetLastError
ERR_TOO_LONG_FILENAME	Too long file name	GetLastError
ERR_TOO_MANY_FILES	More than 64 files cannot be opened at the same time	GetLastError
ERR_TOO_MANY_FORMATTERS	Amount of format specifiers more than the parameters	GetLastError
ERR_TOO_MANY_PARAMETERS	Amount of parameters more than the format specifiers	GetLastError
ERR_TRADE_DEAL_NOT_FOUND	Deal not found	GetLastError
ERR_TRADE_DISABLED	Trading by Expert Advisors prohibited	GetLastError
ERR_TRADE_ORDER_NOT_FOUND	Order not found	GetLastError
ERR_TRADE_POSITION_NOT_FOUND	Position not found	GetLastError
ERR_TRADE_SEND_FAILED	Trade request sending failed	GetLastError
ERR_TRADE_WRONG_PROPERTY	Wrong trade property ID	GetLastError
ERR_USER_ERROR_FIRST	User defined errors start with this code	GetLastError
ERR_WEBREQUEST_CONNECT_FAILED	Failed to connect to specified URL	GetLastError
ERR_WEBREQUEST_INVALID_ADDRESS	Invalid URL	GetLastError

ERR_WEBREQUEST_REQUEST_FAILED	HTTP request failed	GetLastError
ERR_WEBREQUEST_TIMEOUT	Timeout exceeded	GetLastError
ERR_WRONG_DIRECTORYNAME	Wrong directory name	GetLastError
ERR_WRONG_FILEHANDLE	Wrong file handle	GetLastError
ERR_WRONG_FILENAME	Invalid file name	GetLastError
ERR_WRONG_FORMATSTRING	Invalid format string	GetLastError
ERR_WRONG_INTERNAL_PARAMETER	Wrong parameter in the inner call of the client terminal function	GetLastError
ERR_WRONG_STRING_DATE	Wrong date in the string	GetLastError
ERR_WRONG_STRING_OBJECT	Damaged string object	GetLastError
ERR_WRONG_STRING_PARAMETER	Damaged parameter of string type	GetLastError
ERR_WRONG_STRING_TIME	Wrong time in the string	GetLastError
ERR_ZEROSIZE_ARRAY	An array of zero length	GetLastError
FILE_ACCESS_DATE	Date of the last access to the file	FileGetInteger
FILE_ANSI	Strings of ANSI type (one byte symbols). Flag is used in FileOpen() .	FileOpen
FILE_BIN	Binary read/write mode (without string to string conversion). Flag is used in FileOpen() .	FileOpen
FILE_COMMON	The file path in the common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen() , FileCopy() , FileMove() and in FileExists() functions.	FileOpen , FileCopy , FileMove , FileExists
FILE_CREATE_DATE	Date of creation	FileGetInteger
FILE_CSV	CSV file (all its elements are converted to strings of the appropriate type, Unicode or ANSI, and separated by separator). Flag is used in FileOpen() .	FileOpen

FILE_END	Get the end of file sign	FileGetInteger
FILE_EXISTS	Check the existence	FileGetInteger
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)	FileGetInteger
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)	FileGetInteger
FILE_IS_COMMON	The file is opened in a shared folder of all terminals (see FILE_COMMON)	FileGetInteger
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)	FileGetInteger
FILE_IS_READABLE	The opened file is readable (see FILE_READ)	FileGetInteger
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)	FileGetInteger
FILE_IS_WRITABLE	The opened file is writable (see FILE_WRITE)	FileGetInteger
FILE_LINE_END	Get the end of line sign	FileGetInteger
FILE_MODIFY_DATE	Date of the last modification	FileGetInteger
FILE_POSITION	Position of a pointer in the file	FileGetInteger
FILE_READ	File is opened for reading. Flag is used in FileOpen() . When opening a file specification of FILE_WRITE and/or FILE_READ is required.	FileOpen
FILE_REWRITE	Possibility for the file rewrite using functions FileCopy() and FileMove() . The file should exist or should be opened for writing, otherwise the file will not be opened.	FileCopy , FileMove
FILE_SHARE_READ	Shared access for reading from several programs. Flag is used in FileOpen() , but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	FileOpen
FILE_SHARE_WRITE	Shared access for writing from several programs. Flag is used in FileOpen() , but it does not	FileOpen

	replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	
FILE_SIZE	File size in bytes	FileGetInteger
FILE_TXT	Simple text file (the same as csv file, but without taking into account the separators). Flag is used in FileOpen() .	FileOpen
FILE_UNICODE	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen() .	FileOpen
FILE_WRITE	File is opened for writing. Flag is used in FileOpen() . When opening a file specification of FILE_WRITE and/or FILE_READ is required.	FileOpen
FLT_DIG	Number of significant decimal digits for float type	Numerical Type Constants
FLT_EPSILON	Minimal value, which satisfies the condition: $1.0 + \text{DBL_EPSILON} \neq 1.0$ (for float type)	Numerical Type Constants
FLT_MANT_DIG	Bits count in a mantissa for float type	Numerical Type Constants
FLT_MAX	Maximal value, which can be represented by float type	Numerical Type Constants
FLT_MAX_10_EXP	Maximal decimal value of exponent degree for float type	Numerical Type Constants
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	Numerical Type Constants
FLT_MIN	Minimal positive value, which can be represented by float type	Numerical Type Constants
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	Numerical Type Constants
FLT_MIN_EXP	Minimal binary value of exponent degree for float type	Numerical Type Constants
FRIDAY	Friday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade

GANN_DOWN_TREND	Line corresponding to the downward trend	ObjectSetInteger , ObjectGetInteger
GANN_UP_TREND	Line corresponding to the uptrend line	ObjectSetInteger , ObjectGetInteger
GATORJAW_LINE	Jaw line	Indicators Lines
GATORLIPS_LINE	Lips line	Indicators Lines
GATORTEETH_LINE	Teeth line	Indicators Lines
IDABORT	"Abort" button has been pressed	MessageBox
IDCANCEL	"Cancel" button has been pressed	MessageBox
IDCONTINUE	"Continue" button has been pressed	MessageBox
IDIGNORE	"Ignore" button has been pressed	MessageBox
IDNO	"No" button has been pressed	MessageBox
IDOK	"OK" button has been pressed	MessageBox
IDRETRY	"Retry" button has been pressed	MessageBox
IDTRYAGAIN	"Try Again" button has been pressed	MessageBox
IDYES	"Yes" button has been pressed	MessageBox
IND_AC	Accelerator Oscillator	IndicatorCreate , IndicatorParameters
IND_AD	Accumulation/Distribution	IndicatorCreate , IndicatorParameters
IND_ADX	Average Directional Index	IndicatorCreate , IndicatorParameters
IND_ADXW	ADX by Welles Wilder	IndicatorCreate , IndicatorParameters
IND_ALLIGATOR	Alligator	IndicatorCreate , IndicatorParameters
IND_AMA	Adaptive Moving Average	IndicatorCreate , IndicatorParameters
IND_AO	Awesome Oscillator	IndicatorCreate , IndicatorParameters

IND_ATR	Average True Range	IndicatorCreate, IndicatorParameters
IND_BANDS	Bollinger Bands®	IndicatorCreate, IndicatorParameters
IND_BEARS	Bears Power	IndicatorCreate, IndicatorParameters
IND_BULLS	Bulls Power	IndicatorCreate, IndicatorParameters
IND_BWMFI	Market Facilitation Index	IndicatorCreate, IndicatorParameters
IND_CCI	Commodity Channel Index	IndicatorCreate, IndicatorParameters
IND_CHAIKIN	Chaikin Oscillator	IndicatorCreate, IndicatorParameters
IND_CUSTOM	Custom indicator	IndicatorCreate, IndicatorParameters
IND_DEMA	Double Exponential Moving Average	IndicatorCreate, IndicatorParameters
IND_DEMARKER	DeMarker	IndicatorCreate, IndicatorParameters
IND_ENVELOPES	Envelopes	IndicatorCreate, IndicatorParameters
IND_FORCE	Force Index	IndicatorCreate, IndicatorParameters
IND_FRACTALS	Fractals	IndicatorCreate, IndicatorParameters
IND_FRAMA	Fractal Adaptive Moving Average	IndicatorCreate, IndicatorParameters
IND_GATOR	Gator Oscillator	IndicatorCreate, IndicatorParameters
IND_ICHIMOKU	Ichimoku Kinko Hyo	IndicatorCreate, IndicatorParameters
IND_MA	Moving Average	IndicatorCreate, IndicatorParameters
IND_MACD	MACD	IndicatorCreate, IndicatorParameters
IND_MFI	Money Flow Index	IndicatorCreate, IndicatorParameters

IND_MOMENTUM	Momentum	IndicatorCreate, IndicatorParameters
IND_OBV	On Balance Volume	IndicatorCreate, IndicatorParameters
IND_OSMA	OsMA	IndicatorCreate, IndicatorParameters
IND_RSI	Relative Strength Index	IndicatorCreate, IndicatorParameters
IND_RVI	Relative Vigor Index	IndicatorCreate, IndicatorParameters
IND_SAR	Parabolic SAR	IndicatorCreate, IndicatorParameters
IND_STDDEV	Standard Deviation	IndicatorCreate, IndicatorParameters
IND_STOCHASTIC	Stochastic Oscillator	IndicatorCreate, IndicatorParameters
IND_TEMA	Triple Exponential Moving Average	IndicatorCreate, IndicatorParameters
IND_TRIX	Triple Exponential Moving Averages Oscillator	IndicatorCreate, IndicatorParameters
IND_VIDYA	Variable Index Dynamic Average	IndicatorCreate, IndicatorParameters
IND_VOLUMES	Volumes	IndicatorCreate, IndicatorParameters
IND_WPR	Williams' Percent Range	IndicatorCreate, IndicatorParameters
INDICATOR_CALCULATIONS	Auxiliary buffers for intermediate calculations	SetIndexBuffer
INDICATOR_COLOR_INDEX	Color	SetIndexBuffer
INDICATOR_DATA	Data to draw	SetIndexBuffer
INDICATOR_DIGITS	Accuracy of drawing of indicator values	IndicatorSetInteger
INDICATOR_HEIGHT	Fixed height of the indicator's window (the preprocessor command #property indicator_height)	IndicatorSetInteger
INDICATOR_LEVELCOLOR	Color of the level line	IndicatorSetInteger
INDICATOR_LEVELS	Number of levels in the indicator window	IndicatorSetInteger

INDICATOR_LEVELSTYLE	Style of the level line	IndicatorSetInteger
INDICATOR_LEVELTEXT	Level description	IndicatorSetString
INDICATOR_LEVELVALUE	Level value	IndicatorSetDouble
INDICATOR_LEVELWIDTH	Thickness of the level line	IndicatorSetInteger
INDICATOR_MAXIMUM	Maximum of the indicator window	IndicatorSetDouble
INDICATOR_MINIMUM	Minimum of the indicator window	IndicatorSetDouble
INDICATOR_SHORTNAME	Short indicator name	IndicatorSetString
INT_MAX	Maximal value, which can be represented by int type	Numerical Type Constants
INT_MIN	Minimal value, which can be represented by int type	Numerical Type Constants
INVALID_HANDLE	Incorrect handle	Other Constants
IS_DEBUG_MODE	Flag that a mq5-program operates in debug mode	Other Constants
IS_PROFILE_MODE	Flag that a mq5-program operates in profiling mode	Other Constants
KIJUNSEN_LINE	Kijun-sen line	Indicators Lines
LICENSE_DEMO	A trial version of a paid product from the Market. It works only in the strategy tester	MQLInfoInteger
LICENSE_FREE	A free unlimited version	MQLInfoInteger
LICENSE_FULL	A purchased licensed version allows at least 5 activations. The number of activations is specified by seller. Seller may increase the allowed number of activations	MQLInfoInteger
LICENSE_TIME	A version with a limited term license	MQLInfoInteger
LONG_MAX	Maximal value, which can be represented by long type	Numerical Type Constants
LONG_MIN	Minimal value, which can be represented by long type	Numerical Type Constants
LOWER_BAND	Lower limit	Indicators Lines
LOWER_HISTOGRAM	Bottom histogram	Indicators Lines

LOWER_LINE	Bottom line	Indicators Lines
M_1_PI	1/pi	Mathematical Constants
M_2_PI	2/pi	Mathematical Constants
M_2_SQRTPI	2/sqrt(pi)	Mathematical Constants
M_E	e	Mathematical Constants
M_LN10	ln(10)	Mathematical Constants
M_LN2	ln(2)	Mathematical Constants
M_LOG10E	log10(e)	Mathematical Constants
M_LOG2E	log2(e)	Mathematical Constants
M_PI	pi	Mathematical Constants
M_PI_2	pi/2	Mathematical Constants
M_PI_4	pi/4	Mathematical Constants
M_SQRT1_2	1/sqrt(2)	Mathematical Constants
M_SQRT2	sqrt(2)	Mathematical Constants
MAIN_LINE	Main line	Indicators Lines
MB_ABORTRETRYIGNORE	Message window contains three buttons: Abort, Retry and Ignore	MessageBox
MB_CANCELTRYCONTINUE	Message window contains three buttons: Cancel, Try Again, Continue	MessageBox
MB_DEFBUTTON1	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified	MessageBox
MB_DEFBUTTON2	The second button is default	MessageBox
MB_DEFBUTTON3	The third button is default	MessageBox
MB_DEFBUTTON4	The fourth button is default	MessageBox
MB_ICONEXCLAMATION, MB_ICONWARNING	The exclamation/warning sign icon	MessageBox
MB_ICONINFORMATION, MB_ICONASTERISK	The encircled i sign	MessageBox
MB_ICONQUESTION	The question sign icon	MessageBox

MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	The STOP sign icon	MessageBox
MB_OK	Message window contains only one button: OK. Default	MessageBox
MB_OKCANCEL	Message window contains two buttons: OK and Cancel	MessageBox
MB_RETRYCANCEL	Message window contains two buttons: Retry and Cancel	MessageBox
MB_YESNO	Message window contains two buttons: Yes and No	MessageBox
MB_YESNOCANCEL	Message window contains three buttons: Yes, No and Cancel	MessageBox
MINUSDI_LINE	Line -DI	Indicators Lines
MODE_EMA	Exponential averaging	Smoothing Methods
MODE_LWMA	Linear-weighted averaging	Smoothing Methods
MODE_SMA	Simple averaging	Smoothing Methods
MODE_SMMA	Smoothed averaging	Smoothing Methods
MONDAY	Monday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
MQL_DEBUG	The flag, that indicates the debug mode	MQLInfoInteger
MQL_DLLS_ALLOWED	The permission to use DLL for the given executed program	MQLInfoInteger
MQL_FRAME_MODE	The flag, that indicates the Expert Advisor operating in gathering optimization result frames mode	MQLInfoInteger
MQL_LICENSE_TYPE	Type of license of the EX5 module. The license refers to the EX5 module, from which a request is made using <code>MQLInfoInteger(MQL_LICENSE_TYPE)</code> .	MQLInfoInteger
MQL_MEMORY_LIMIT	Maximum possible amount of dynamic memory for MQL5 program in MB	MQLInfoInteger

MQL_MEMORY_USED	The memory size used by MQL5 program in MB	MQLInfoInteger
MQL_OPTIMIZATION	The flag, that indicates the optimization process	MQLInfoInteger
MQL_PROFILER	The flag, that indicates the program operating in the code profiling mode	MQLInfoInteger
MQL_PROGRAM_NAME	Name of the mql5-program executed	MQLInfoString
MQL_PROGRAM_PATH	Path for the given executed program	MQLInfoString
MQL_PROGRAM_TYPE	Type of the mql5 program	MQLInfoInteger
MQL_SIGNALS_ALLOWED	The permission to modify the Signals for the given executed program	MQLInfoInteger
MQL_TESTER	The flag, that indicates the tester process	MQLInfoInteger
MQL_TRADE_ALLOWED	The permission to trade for the given executed program	MQLInfoInteger
MQL_VISUAL_MODE	The flag, that indicates the visual tester process	MQLInfoInteger
NULL	Zero for any types	Other Constants
OBJ_ALL_PERIODS	The object is drawn in all timeframes	ObjectSetInteger , ObjectGetInteger
OBJ_ARROW	Arrow	Object Types
OBJ_ARROW_BUY	Buy Sign	Object Types
OBJ_ARROW_CHECK	Check Sign	Object Types
OBJ_ARROW_DOWN	Arrow Down	Object Types
OBJ_ARROW_LEFT_PRICE	Left Price Label	Object Types
OBJ_ARROW_RIGHT_PRICE	Right Price Label	Object Types
OBJ_ARROW_SELL	Sell Sign	Object Types
OBJ_ARROW_STOP	Stop Sign	Object Types
OBJ_ARROW_THUMB_DOWN	Thumbs Down	Object Types
OBJ_ARROW_THUMB_UP	Thumbs Up	Object Types
OBJ_ARROW_UP	Arrow Up	Object Types
OBJ_ARROWED_LINE	Arrowed Line	Object Types

<u>OBJ_BITMAP</u>	Bitmap	<u>Object Types</u>
<u>OBJ_BITMAP_LABEL</u>	Bitmap Label	<u>Object Types</u>
<u>OBJ_BUTTON</u>	Button	<u>Object Types</u>
<u>OBJ_CHANNEL</u>	Equidistant Channel	<u>Object Types</u>
<u>OBJ_CHART</u>	Chart	<u>Object Types</u>
<u>OBJ_CYCLES</u>	Cycle Lines	<u>Object Types</u>
<u>OBJ_EDIT</u>	Edit	<u>Object Types</u>
<u>OBJ_ELLIOTWAVE3</u>	Elliott Correction Wave	<u>Object Types</u>
<u>OBJ_ELLIOTWAVE5</u>	Elliott Motive Wave	<u>Object Types</u>
<u>OBJ_ELLIPSE</u>	Ellipse	<u>Object Types</u>
<u>OBJ_EVENT</u>	The "Event" object corresponding to an event in the economic calendar	<u>Object Types</u>
<u>OBJ_EXPANSION</u>	Fibonacci Expansion	<u>Object Types</u>
<u>OBJ_FIBO</u>	Fibonacci Retracement	<u>Object Types</u>
<u>OBJ_FIBOARC</u>	Fibonacci Arcs	<u>Object Types</u>
<u>OBJ_FIBOCHANNEL</u>	Fibonacci Channel	<u>Object Types</u>
<u>OBJ_FIBOFAN</u>	Fibonacci Fan	<u>Object Types</u>
<u>OBJ_FIBOTIMES</u>	Fibonacci Time Zones	<u>Object Types</u>
<u>OBJ_GANNFAN</u>	Gann Fan	<u>Object Types</u>
<u>OBJ_GANNGRID</u>	Gann Grid	<u>Object Types</u>
<u>OBJ_GANNLINE</u>	Gann Line	<u>Object Types</u>
<u>OBJ_HLINE</u>	Horizontal Line	<u>Object Types</u>
<u>OBJ_LABEL</u>	Label	<u>Object Types</u>
<u>OBJ_NO_PERIODS</u>	The object is not drawn in all timeframes	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
<u>OBJ_PERIOD_D1</u>	The object is drawn in day charts	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
<u>OBJ_PERIOD_H1</u>	The object is drawn in 1-hour chart	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
<u>OBJ_PERIOD_H12</u>	The object is drawn in 12-hour chart	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
<u>OBJ_PERIOD_H2</u>	The object is drawn in 2-hour chart	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>

OBJ_PERIOD_H3	The object is drawn in 3-hour chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_H4	The object is drawn in 4-hour chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_H6	The object is drawn in 6-hour chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_H8	The object is drawn in 8-hour chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M1	The object is drawn in 1-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M10	The object is drawn in 10-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M12	The object is drawn in 12-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M15	The object is drawn in 15-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M2	The object is drawn in 2-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M20	The object is drawn in 20-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M3	The object is drawn in 3-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M30	The object is drawn in 30-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M4	The object is drawn in 4-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M5	The object is drawn in 5-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_M6	The object is drawn in 6-minute chart	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_MN1	The object is drawn in month charts	ObjectSetInteger , ObjectGetInteger
OBJ_PERIOD_W1	The object is drawn in week charts	ObjectSetInteger , ObjectGetInteger
OBJ_PITCHFORK	Andrews' Pitchfork	Object Types
OBJ_RECTANGLE	Rectangle	Object Types
OBJ_RECTANGLE_LABEL	The "Rectangle label" object for creating and designing the custom graphical interface.	Object Types

<u>OBJ_REGRESSION</u>	Linear Regression Channel	<u>Object Types</u>
<u>OBJ_STDDEVCHANNEL</u>	Standard Deviation Channel	<u>Object Types</u>
<u>OBJ_TEXT</u>	Text	<u>Object Types</u>
<u>OBJ_TREND</u>	Trend Line	<u>Object Types</u>
<u>OBJ_TRENDBYANGLE</u>	Trend Line By Angle	<u>Object Types</u>
<u>OBJ_TRIANGLE</u>	Triangle	<u>Object Types</u>
<u>OBJ_VLINE</u>	Vertical Line	<u>Object Types</u>
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_ANCHOR	Location of the anchor point of a graphical object	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_ANGLE	Angle. For the objects with no angle specified, created from a program, the value is equal to <u>EMPTY_VALUE</u>	<u>ObjectSetDouble</u> , <u>ObjectGetDouble</u>
OBJPROP_ARROWCODE	Arrow code for the Arrow object	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_BACK	Object in the background	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also <u>Resources</u>	<u>ObjectSetString</u> , <u>ObjectGetString</u>
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_CHART_ID	ID of the "Chart" object (<u>OBJ_CHART</u>). It allows working with the properties of this object like with a normal chart using the functions described in <u>Chart Operations</u> , but there some <u>exceptions</u> .	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>
OBJPROP_CHART_SCALE	The scale for the Chart object	<u>ObjectSetInteger</u> , <u>ObjectGetInteger</u>

OBJPROP_COLOR	Color	ObjectSetInteger , ObjectGetInteger
OBJPROP_CORNER	The corner of the chart to link a graphical object	ObjectSetInteger , ObjectGetInteger
OBJPROP_CREATETIME	Time of object creation	ObjectSetInteger , ObjectGetInteger
OBJPROP_DATE_SCALE	Displaying the time scale for the Chart object	ObjectSetInteger , ObjectGetInteger
OBJPROP_DEGREE	Level of the Elliott Wave Marking	ObjectSetInteger , ObjectGetInteger
OBJPROP_DEVIATION	Deviation for the Standard Deviation Channel	ObjectSetDouble , ObjectGetDouble
OBJPROP_DIRECTION	Trend of the Gann object	ObjectSetInteger , ObjectGetInteger
OBJPROP_DRAWLINES	Displaying lines for marking the Elliott Wave	ObjectSetInteger , ObjectGetInteger
OBJPROP_ELLIPSE	Showing the full ellipse of the Fibonacci Arc object (OBJ_FIBOARC)	ObjectSetInteger , ObjectGetInteger
OBJPROP_FILL	Fill an object with color (for OBJ_RECTANGLE , OBJ_TRIANGLE , OBJ_ELLIPSE , OBJ_CHANNEL , OBJ_STDDEVCHANNEL , OBJ_REGRESSION)	ObjectSetInteger , ObjectGetInteger
OBJPROP_FONT	Font	ObjectSetString , ObjectGetString
OBJPROP_FONTSIZE	Font size	ObjectSetInteger , ObjectGetInteger
OBJPROP_HIDDEN	Prohibit showing of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects". The true value allows to hide an object from the list. By default, true is set to the objects that display calendar events, trading history and to the objects created from MQL5 programs . To see such graphical objects and access their properties, click on the	ObjectSetInteger , ObjectGetInteger

	"All" button in the "List of objects" window.	
OBJPROP_LEVELCOLOR	Color of the line-level	ObjectSetInteger , ObjectGetInteger
OBJPROP_LEVELS	Number of levels	ObjectSetInteger , ObjectGetInteger
OBJPROP_LEVELSTYLE	Style of the line-level	ObjectSetInteger , ObjectGetInteger
OBJPROP_LEVELTEXT	Level description	ObjectSetString , ObjectGetString
OBJPROP_LEVELVALUE	Level value	ObjectSetDouble , ObjectGetDouble
OBJPROP_LEVELWIDTH	Thickness of the line-level	ObjectSetInteger , ObjectGetInteger
OBJPROP_NAME	Object name	ObjectSetString , ObjectGetString
OBJPROP_PERIOD	Timeframe for the Chart object	ObjectSetInteger , ObjectGetInteger
OBJPROP_PRICE	Price coordinate	ObjectSetDouble , ObjectGetDouble
OBJPROP_PRICE_SCALE	Displaying the price scale for the Chart object	ObjectSetInteger , ObjectGetInteger
OBJPROP_RAY	A vertical line goes through all the windows of a chart	ObjectSetInteger , ObjectGetInteger
OBJPROP_RAY_LEFT	Ray goes to the left	ObjectSetInteger , ObjectGetInteger
OBJPROP_RAY_RIGHT	Ray goes to the right	ObjectSetInteger , ObjectGetInteger
OBJPROP_READONLY	Ability to edit text in the Edit object	ObjectSetInteger , ObjectGetInteger
OBJPROP_SCALE	Scale (properties of Gann objects and Fibonacci Arcs)	ObjectSetDouble , ObjectGetDouble
OBJPROP_SELECTABLE	Object availability	ObjectSetInteger , ObjectGetInteger
OBJPROP_SELECTED	Object is selected	ObjectSetInteger , ObjectGetInteger
OBJPROP_STATE	Button state (pressed / depressed)	ObjectSetInteger , ObjectGetInteger

OBJPROP_STYLE	Style	ObjectSetInteger , ObjectGetInteger
OBJPROP_SYMBOL	Symbol for the Chart object	ObjectSetString , ObjectGetString
OBJPROP_TEXT	Description of the object (the text contained in the object)	ObjectSetString , ObjectGetString
OBJPROP_TIME	Time coordinate	ObjectSetInteger , ObjectGetInteger
OBJPROP_TIMEFRAMES	Visibility of an object at timeframes	ObjectSetInteger , ObjectGetInteger
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value to it	ObjectSetString , ObjectGetString
OBJPROP_TYPE	Object type	ObjectSetInteger , ObjectGetInteger
OBJPROP_WIDTH	Line thickness	ObjectSetInteger , ObjectGetInteger
OBJPROP_XDISTANCE	The distance in pixels along the X axis from the binding corner (see note)	ObjectSetInteger , ObjectGetInteger
OBJPROP_XOFFSET	The X coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	ObjectSetInteger , ObjectGetInteger
OBJPROP_XSIZE	The object's width along the X axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	ObjectSetInteger , ObjectGetInteger

OBJPROP_YDISTANCE	The distance in pixels along the Y axis from the binding corner (see note)	ObjectSetInteger , ObjectGetInteger
OBJPROP_YOFFSET	The Y coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	ObjectSetInteger , ObjectGetInteger
OBJPROP_YSIZE	The object's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	ObjectSetInteger , ObjectGetInteger
OBJPROP_ZORDER	Priority of a graphical object for receiving events of clicking on a chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When applying objects one over another, only one of them with the highest priority will receive the CHARTEVENT_CLICK event.	ObjectSetInteger , ObjectGetInteger
ORDER_COMMENT	Order comment	OrderGetString , HistoryOrderGetString
ORDER_FILLING_FOK	This filling policy means that an order can be filled only in the specified amount. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The required volume can be filled using several offers available on the market at the moment.	OrderGetInteger , HistoryOrderGetInteger

ORDER_FILLING_IOC	This mode means that a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case the the entire volume of an order cannot be filled, the available volume of it will be filled, and the remaining volume will be canceled.	OrderGetInteger , HistoryOrderGetInteger
ORDER_FILLING_RETURN	This policy is used only for market orders (ORDER_TYPE_BUY and ORDER_TYPE_SELL), limit and stop limit orders (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT) and only for the symbols with Market or Exchange execution . In case of partial filling a market or limit order with remaining volume is not canceled but processed further. For the activation of the ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT orders, a corresponding limit order ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT with the ORDER_FILLING_RETURN execution type is created.	OrderGetInteger , HistoryOrderGetInteger
ORDER_MAGIC	ID of an Expert Advisor that has placed the order (designed to ensure that each Expert Advisor places its own unique number)	OrderGetInteger , HistoryOrderGetInteger
ORDER_POSITION_ID	Position identifier that is set to an order as soon as it is executed. Each executed order results in a deal that opens or modifies an already existing position. The identifier of	OrderGetInteger , HistoryOrderGetInteger

	exactly this position is set to the executed order at this moment.	
ORDER_PRICE_CURRENT	The current price of the order symbol	OrderGetDouble , HistoryOrderGetDouble
ORDER_PRICE_OPEN	Price specified in the order	OrderGetDouble , HistoryOrderGetDouble
ORDER_PRICE_STOPLIMIT	The Limit order price for the StopLimit order	OrderGetDouble , HistoryOrderGetDouble
ORDER_SL	Stop Loss value	OrderGetDouble , HistoryOrderGetDouble
ORDER_STATE	Order state	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_CANCELED	Order canceled by client	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_EXPIRED	Order expired	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_FILLED	Order fully executed	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_PARTIAL	Order partially executed	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_PLACED	Order accepted	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_REJECTED	Order rejected	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_REQUEST_ADD	Order is being registered (placing to the trading system)	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_REQUEST_CANCEL	Order is being deleted (deleting from the trading system)	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_REQUEST_MODIFY	Order is being modified (changing its parameters)	OrderGetInteger , HistoryOrderGetInteger
ORDER_STATE_STARTED	Order checked, but not yet accepted by broker	OrderGetInteger , HistoryOrderGetInteger
ORDER_SYMBOL	Symbol of the order	OrderGetString , HistoryOrderGetString
ORDER_TIME_DAY	Good till current trade day order	OrderGetInteger , HistoryOrderGetInteger

ORDER_TIME_DONE	Order execution or cancellation time	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_DONE_MSC	Order execution/cancellation time in milliseconds since 01.01.1970	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_EXPIRATION	Order expiration time	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_GTC	Good till cancel order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_SETUP	Order setup time	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_SETUP_MSC	The time of placing an order for execution in milliseconds since 01.01.1970	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_SPECIFIED	Good till expired order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TIME_SPECIFIED_DAY	The order will be effective till 23:59:59 of the specified day. If this time is outside a trading session, the order expires in the nearest trading time.	OrderGetInteger , HistoryOrderGetInteger
ORDER_TP	Take Profit value	OrderGetDouble , HistoryOrderGetDouble
ORDER_TYPE	Order type	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_BUY	Market Buy order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_BUY_LIMIT	Buy Limit pending order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_BUY_STOP	Buy Stop pending order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_BUY_STOP_LIMIT	Upon reaching the order price, a pending Buy Limit order is placed at the StopLimit price	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_FILLING	Order filling type	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_SELL	Market Sell order	OrderGetInteger , HistoryOrderGetInteger

ORDER_TYPE_SELL_LIMIT	Sell Limit pending order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_SELL_STOP	Sell Stop pending order	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_SELL_STOP_LIMIT	Upon reaching the order price, a pending Sell Limit order is placed at the StopLimit price	OrderGetInteger , HistoryOrderGetInteger
ORDER_TYPE_TIME	Order lifetime	OrderGetInteger , HistoryOrderGetInteger
ORDER_VOLUME_CURRENT	Order current volume	OrderGetDouble , HistoryOrderGetDouble
ORDER_VOLUME_INITIAL	Order initial volume	OrderGetDouble , HistoryOrderGetDouble
PERIOD_CURRENT	Current timeframe	Chart Timeframes
PERIOD_D1	1 day	Chart Timeframes
PERIOD_H1	1 hour	Chart Timeframes
PERIOD_H12	12 hours	Chart Timeframes
PERIOD_H2	2 hours	Chart Timeframes
PERIOD_H3	3 hours	Chart Timeframes
PERIOD_H4	4 hours	Chart Timeframes
PERIOD_H6	6 hours	Chart Timeframes
PERIOD_H8	8 hours	Chart Timeframes
PERIOD_M1	1 minute	Chart Timeframes
PERIOD_M10	10 minutes	Chart Timeframes
PERIOD_M12	12 minutes	Chart Timeframes
PERIOD_M15	15 minutes	Chart Timeframes
PERIOD_M2	2 minutes	Chart Timeframes
PERIOD_M20	20 minutes	Chart Timeframes
PERIOD_M3	3 minutes	Chart Timeframes
PERIOD_M30	30 minutes	Chart Timeframes
PERIOD_M4	4 minutes	Chart Timeframes
PERIOD_M5	5 minutes	Chart Timeframes
PERIOD_M6	6 minutes	Chart Timeframes
PERIOD_MN1	1 month	Chart Timeframes

PERIOD_W1	1 week	Chart Timeframes
PLOT_ARROW	Arrow code for style DRAW_ARROW	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_ARROW_SHIFT	Vertical shift of arrows for style DRAW_ARROW	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_COLOR_INDEXES	The number of colors	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_DRAW_BEGIN	Number of initial bars without drawing and values in the DataWindow	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_DRAW_TYPE	Type of graphical construction	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_EMPTY_VALUE	An empty value for plotting, for which there is no drawing	PlotIndexSetDouble
PLOT_LABEL	The name of the indicator graphical series to display in the DataWindow. When working with complex graphical styles requiring several indicator buffers for display, the names for each buffer can be specified using ";" as a separator. Sample code is shown in DRAW_CANDLES	PlotIndexSetString
PLOT_LINE_COLOR	The index of a buffer containing the drawing color	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_LINE_STYLE	Drawing line style	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_LINE_WIDTH	The thickness of the drawing line	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_SHIFT	Shift of indicator plotting along the time axis in bars	PlotIndexSetInteger , PlotIndexGetInteger
PLOT_SHOW_DATA	Sign of display of construction values in the DataWindow	PlotIndexSetInteger , PlotIndexGetInteger
PLUSDI_LINE	Line +DI	Indicators Lines
POINTER_AUTOMATIC	Pointer of any objects created automatically (not using new())	CheckPointer
POINTER_DYNAMIC	Pointer of the object created by the new() operator	CheckPointer

POINTER_INVALID	Incorrect pointer	CheckPointer
POSITION_COMMENT	Position comment	PositionGetString
POSITION_COMMISSION	Commission	PositionGetDouble
POSITION_IDENTIFIER	Position identifier is a unique number that is assigned to every newly opened position and doesn't change during the entire lifetime of the position. Position turnover doesn't change its identifier.	PositionGetInteger
POSITION_MAGIC	Position magic number (see ORDER_MAGIC)	PositionGetInteger
POSITION_PRICE_CURRENT	Current price of the position symbol	PositionGetDouble
POSITION_PRICE_OPEN	Position open price	PositionGetDouble
POSITION_PROFIT	Current profit	PositionGetDouble
POSITION_SL	Stop Loss level of opened position	PositionGetDouble
POSITION_SWAP	Cumulative swap	PositionGetDouble
POSITION_SYMBOL	Symbol of the position	PositionGetString
POSITION_TIME	Position open time	PositionGetInteger
POSITION_TIME_MSC	Position opening time in milliseconds since 01.01.1970	PositionGetInteger
POSITION_TIME_UPDATE	Position changing time in seconds since 01.01.1970	PositionGetInteger
POSITION_TIME_UPDATE_MSC	Position changing time in milliseconds since 01.01.1970	PositionGetInteger
POSITION_TP	Take Profit level of opened position	PositionGetDouble
POSITION_TYPE	Position type	PositionGetInteger
POSITION_TYPE_BUY	Buy	PositionGetInteger
POSITION_TYPE_SELL	Sell	PositionGetInteger
POSITION_VOLUME	Position volume	PositionGetDouble
PRICE_CLOSE	Close price	Price Constants
PRICE_HIGH	The maximum price for the period	Price Constants

PRICE_LOW	The minimum price for the period	Price Constants
PRICE_MEDIAN	Median price, (high + low)/2	Price Constants
PRICE_OPEN	Open price	Price Constants
PRICE_TYPICAL	Typical price, (high + low + close)/3	Price Constants
PRICE_WEIGHTED	Average price, (high + low + close + close)/4	Price Constants
PROGRAM_EXPERT	Expert	MQLInfoInteger
PROGRAM_INDICATOR	Indicator	MQLInfoInteger
PROGRAM_SCRIPT	Script	MQLInfoInteger
REASON_ACCOUNT	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings	UninitializeReason , OnDeinit
REASON_CHARTCHANGE	Symbol or chart period has been changed	UninitializeReason , OnDeinit
REASON_CHARTCLOSE	Chart has been closed	UninitializeReason , OnDeinit
REASON_CLOSE	Terminal has been closed	UninitializeReason , OnDeinit
REASON_INITFAILED	This value means that OnInit() handler has returned a nonzero value	UninitializeReason , OnDeinit
REASON_PARAMETERS	Input parameters have been changed by a user	UninitializeReason , OnDeinit
REASON_PROGRAM	Expert Advisor terminated its operation by calling the ExpertRemove() function	UninitializeReason , OnDeinit
REASON_RECOMPILE	Program has been recompiled	UninitializeReason , OnDeinit
REASON_REMOVE	Program has been deleted from the chart	UninitializeReason , OnDeinit
REASON_TEMPLATE	A new template has been applied	UninitializeReason , OnDeinit
SATURDAY	Saturday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
SEEK_CUR	Current position of a file pointer	FileSeek

SEEK_END	File end	FileSeek
SEEK_SET	File beginning	FileSeek
SENKOUSPANA_LINE	Senkou Span A line	Indicators Lines
SENKOUSPANB_LINE	Senkou Span B line	Indicators Lines
SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	SeriesInfoInteger
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	SeriesInfoInteger
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	SeriesInfoInteger
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	SeriesInfoInteger
SERIES_SYNCHRONIZED	Symbol/period data synchronization flag for the current moment	SeriesInfoInteger
SERIES_TERMINAL_FIRSTDATE	The very first date in the history of the symbol in the client terminal, regardless of the timeframe	SeriesInfoInteger
SHORT_MAX	Maximal value, which can be represented by short type	Numerical Type Constants
SHORT_MIN	Minimal value, which can be represented by short type	Numerical Type Constants
SIGNAL_BASE_AUTHOR_LOGIN	Author login	SignalBaseGetString
SIGNAL_BASE_BALANCE	Account balance	SignalBaseGetDouble
SIGNAL_BASE_BROKER	Broker name (company)	SignalBaseGetString
SIGNAL_BASE_BROKER_SERVER	Broker server	SignalBaseGetString
SIGNAL_BASE_CURRENCY	Signal base currency	SignalBaseGetString
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)	SignalBaseGetInteger
SIGNAL_BASE_DATE_STARTED	Monitoring starting date	SignalBaseGetInteger
SIGNAL_BASE_EQUITY	Account equity	SignalBaseGetDouble
SIGNAL_BASE_GAIN	Account gain	SignalBaseGetDouble

SIGNAL_BASE_ID	Signal ID	SignalBaseGetInteger
SIGNAL_BASE_LEVERAGE	Account leverage	SignalBaseGetInteger
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown	SignalBaseGetDouble
SIGNAL_BASE_NAME	Signal name	SignalBaseGetString
SIGNAL_BASE_PIPS	Profit in pips	SignalBaseGetInteger
SIGNAL_BASE_PRICE	Signal subscription price	SignalBaseGetDouble
SIGNAL_BASE_RATING	Position in rating	SignalBaseGetInteger
SIGNAL_BASE_ROI	Return on Investment (%)	SignalBaseGetDouble
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers	SignalBaseGetInteger
SIGNAL_BASE_TRADE_MODE	Account type (0-real, 1-demo, 2-contest)	SignalBaseGetInteger
SIGNAL_BASE_TRADES	Number of trades	SignalBaseGetInteger
SIGNAL_INFO_CONFIRMATION_S_DISABLED	The flag enables synchronization without confirmation dialog	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_DEPOSIT_PERCENT	Deposit percent (%)	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_EQUITY_LIMIT	Equity limit	SignalInfoGetDouble , SignalInfoSetDouble
SIGNAL_INFO_ID	Signal id, r/o	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_NAME	Signal name, r/o	SignalInfoGetString
SIGNAL_INFO_SLIPPAGE	Slippage (used when placing market orders in synchronization of positions and copying of trades)	SignalInfoGetDouble , SignalInfoSetDouble
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o	SignalInfoGetInteger , SignalInfoSetInteger
SIGNAL_INFO_VOLUME_PERCENT	Maximum percent of deposit used (%), r/o	SignalInfoGetDouble , SignalInfoSetDouble
SIGNAL_LINE	Signal line	Indicators Lines

STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous drawdowns; here the largest value is taken	TesterStatistics
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	TesterStatistics
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	TesterStatistics
STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	TesterStatistics
STAT_BALANCEMIN	Minimum balance value	TesterStatistics
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	TesterStatistics
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	TesterStatistics
STAT_CONPROFITMAX	Maximum profit in a series of profitable trades. The value is greater than or equal to zero	TesterStatistics
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	TesterStatistics

STAT_CUSTOM_ONTESTER	The value of the calculated custom optimization criterion returned by the OnTester() function	TesterStatistics
STAT_DEALS	The number of deals	TesterStatistics
STAT_EQUITY_DD	Maximum equity drawdown in monetary terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	TesterStatistics
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	TesterStatistics
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	TesterStatistics
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity drawdown in monetary terms (STAT_EQUITY_DD).	TesterStatistics
STAT_EQUITYMIN	Minimum equity value	TesterStatistics
STAT_EXPECTED_PAYOFF	Expected payoff	TesterStatistics
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	TesterStatistics
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal to zero	TesterStatistics
STAT_INITIAL_DEPOSIT	The value of the initial deposit	TesterStatistics
STAT_LONG_TRADES	Long trades	TesterStatistics
STAT_LOSS_TRADES	Losing trades	TesterStatistics

STAT_LOSSTRADES_AVGCON	Average length of a losing series of trades	TesterStatistics
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	TesterStatistics
STAT_MAX_CONLOSSES	The total loss of the longest series of losing trades	TesterStatistics
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	TesterStatistics
STAT_MAX_CONWINS	The total profit of the longest series of profitable trades	TesterStatistics
STAT_MAX_LOSSTRADE	Maximum loss - the lowest value of all losing trades. The value is less than or equal to zero	TesterStatistics
STAT_MAX_PROFITTRADE	Maximum profit - the largest value of all profitable trades. The value is greater than or equal to zero	TesterStatistics
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	TesterStatistics
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS (STAT_GROSS_LOSS is always less than or equal to zero)	TesterStatistics
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of STAT_GROSS_PROFIT/STAT_GROSS_LOSS. If STAT_GROSS_LOSS=0, the profit factor is equal to DBL_MAX	TesterStatistics
STAT_PROFIT_LONGTRADES	Profitable long trades	TesterStatistics
STAT_PROFIT_SHORTTRADES	Profitable short trades	TesterStatistics
STAT_PROFIT_TRADES	Profitable trades	TesterStatistics
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	TesterStatistics
STAT_RECOVERY_FACTOR	Recovery factor, equal to the ratio of	TesterStatistics

	STAT_PROFIT/STAT_BALANCE_DD	
STAT_SHARPE_RATIO	Sharpe ratio	TesterStatistics
STAT_SHORT_TRADES	Short trades	TesterStatistics
STAT_TRADES	The number of trades	TesterStatistics
STAT_WITHDRAWAL	Money withdrawn from an account	TesterStatistics
STO_CLOSECLOSE	Calculation is based on Close/Close prices	Price Constants
STO_LOWHIGH	Calculation is based on Low/High prices	Price Constants
STYLE_DASH	Broken line	Drawing Styles
STYLE_DASHDOT	Dash-dot line	Drawing Styles
STYLE_DASHDOTDOT	Dash - two points	Drawing Styles
STYLE_DOT	Dotted line	Drawing Styles
STYLE_SOLID	Solid line	Drawing Styles
SUNDAY	Sunday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
SYMBOL_ASK	Ask - best buy offer	SymbolInfoDouble
SYMBOL_ASKHIGH	Maximal Ask of the day	SymbolInfoDouble
SYMBOL_ASKLOW	Minimal Ask of the day	SymbolInfoDouble
SYMBOL_BANK	Feeder of the current quote	SymbolInfoString
SYMBOL_BASIS	The underlying asset of a derivative	SymbolInfoString
SYMBOL_BID	Bid - best sell offer	SymbolInfoDouble
SYMBOL_BIDHIGH	Maximal Bid of the day	SymbolInfoDouble
SYMBOL_BIDLOW	Minimal Bid of the day	SymbolInfoDouble
SYMBOL_CALC_MODE_CFD	CFD mode - calculation of margin and profit for CFD	SymbolInfoInteger
SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - calculation of margin and profit for CFD by indexes	SymbolInfoInteger
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - calculation of margin and	SymbolInfoInteger

	profit for CFD at leverage trading	
SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - calculation of margin and profit for trading futures contracts on a stock exchange	SymbolInfoInteger
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode - calculation of margin and profit for trading futures contracts on FORTS.	SymbolInfoInteger
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode - calculation of margin and profit for trading securities on a stock exchange	SymbolInfoInteger
SYMBOL_CALC_MODE_FOREX	Forex mode - calculation of profit and margin for Forex	SymbolInfoInteger
SYMBOL_CALC_MODE_FUTURES	Futures mode - calculation of margin and profit for futures	SymbolInfoInteger
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - a symbol is used as a non-tradable asset on a trading account.	SymbolInfoInteger
SYMBOL_CURRENCY_BASE	Basic currency of a symbol	SymbolInfoString
SYMBOL_CURRENCY_MARGIN	Margin currency	SymbolInfoString
SYMBOL_CURRENCY_PROFIT	Profit currency	SymbolInfoString
SYMBOL_DESCRIPTION	Symbol description	SymbolInfoString
SYMBOL_DIGITS	Digits after a decimal point	SymbolInfoInteger
SYMBOL_EXPIRATION_DAY	The order is valid till the end of the day	SymbolInfoInteger
SYMBOL_EXPIRATION_GTC	The order is valid during the unlimited time period, until it is explicitly canceled	SymbolInfoInteger
SYMBOL_EXPIRATION_MODE	Flags of allowed order expiration modes	SymbolInfoInteger
SYMBOL_EXPIRATION_SPECIFIED	The expiration time is specified in the order	SymbolInfoInteger
SYMBOL_EXPIRATION_SPECIFIED_DAY	The expiration date is specified in the order	SymbolInfoInteger
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	SymbolInfoInteger

SYMBOL_FILLING_FOK	This policy means that a deal can be executed only with the specified volume. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The required volume can be filled using several offers available on the market at the moment.	SymbolInfoInteger
SYMBOL_FILLING_IOC	In this case a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case the order cannot be filled completely, the available volume of the order will be filled, and the remaining volume will be canceled. The possibility of using IOC orders is determined at the trade server.	SymbolInfoInteger
SYMBOL_FILLING_MODE	Flags of allowed order filling modes	SymbolInfoInteger
SYMBOL_ISIN	The name of a symbol in the ISIN system (International Securities Identification Number). The International Securities Identification Number is a 12-digit alphanumeric code that uniquely identifies a security. The presence of this symbol property is determined on the side of a trade server.	SymbolInfoString
SYMBOL_LAST	Price of the last deal	SymbolInfoDouble
SYMBOL_LASTHIGH	Maximal Last of the day	SymbolInfoDouble
SYMBOL_LASTLOW	Minimal Last of the day	SymbolInfoDouble
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening a position with the volume of one lot. It is used for checking a client's	SymbolInfoDouble

	assets when he or she enters the market.	
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency of the symbol, charged from one lot. It is used for checking a client's assets when his/her account state changes. If the maintenance margin is equal to 0, the initial margin is used.	SymbolInfoDouble
SYMBOL_OPTION_MODE	Option type	SymbolInfoInteger
SYMBOL_OPTION_MODE_EUROPEAN	European option may only be exercised on a specified date (expiration, execution date, delivery date)	SymbolInfoInteger
SYMBOL_OPTION_MODE_AMERICAN	American option may be exercised on any trading day on or before expiry. The period within which a buyer can exercise the option is specified for it	SymbolInfoInteger
SYMBOL_OPTION_RIGHT	Option right (Call/Put)	SymbolInfoInteger
SYMBOL_OPTION_RIGHT_CALL	A call option gives you the right to buy an asset at a specified price	SymbolInfoInteger
SYMBOL_OPTION_RIGHT_PUT	A put option gives you the right to sell an asset at a specified price	SymbolInfoInteger
SYMBOL_OPTION_STRIKE	The strike price of an option. The price at which an option buyer can buy (in a Call option) or sell (in a Put option) the underlying asset, and the option seller is obliged to sell or buy the appropriate amount of the underlying asset.	SymbolInfoDouble
SYMBOL_ORDER_LIMIT	Limit orders are allowed (Buy Limit and Sell Limit)	SymbolInfoInteger
SYMBOL_ORDER_MARKET	Market orders are allowed (Buy and Sell)	SymbolInfoInteger
SYMBOL_ORDER_MODE	Flags of allowed order types	SymbolInfoInteger

SYMBOL_ORDER_SL	Stop Loss is allowed	SymbolInfoInteger
SYMBOL_ORDER_STOP	Stop orders are allowed (Buy Stop and Sell Stop)	SymbolInfoInteger
SYMBOL_ORDER_STOP_LIMIT	Stop-limit orders are allowed (Buy Stop Limit and Sell Stop Limit)	SymbolInfoInteger
SYMBOL_ORDER_TP	Take Profit is allowed	SymbolInfoInteger
SYMBOL_PATH	Path in the symbol tree	SymbolInfoString
SYMBOL_POINT	Symbol point value	SymbolInfoDouble
SYMBOL_SELECT	Symbol is selected in Market Watch	SymbolInfoInteger
SYMBOL_SESSION_AW	Average weighted price of the current session	SymbolInfoDouble
SYMBOL_SESSION_BUY_ORDERS	Number of Buy orders at the moment	SymbolInfoInteger
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Current volume of Buy orders	SymbolInfoDouble
SYMBOL_SESSION_CLOSE	Close price of the current session	SymbolInfoDouble
SYMBOL_SESSION_DEALS	Number of deals in the current session	SymbolInfoInteger
SYMBOL_SESSION_INTEREST	Summary open interest	SymbolInfoDouble
SYMBOL_SESSION_OPEN	Open price of the current session	SymbolInfoDouble
SYMBOL_SESSION_PRICE_LIMIT_MAX	Maximal price of the current session	SymbolInfoDouble
SYMBOL_SESSION_PRICE_LIMIT_MIN	Minimal price of the current session	SymbolInfoDouble
SYMBOL_SESSION_PRICE_SETTLEMENT	Settlement price of the current session	SymbolInfoDouble
SYMBOL_SESSION_SELL_ORDERS	Number of Sell orders at the moment	SymbolInfoInteger
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Current volume of Sell orders	SymbolInfoDouble
SYMBOL_SESSION_TURNOVER	Summary turnover of the current session	SymbolInfoDouble
SYMBOL_SESSION_VOLUME	Summary volume of current session deals	SymbolInfoDouble

SYMBOL_SPREAD	Spread value in points	SymbolInfoInteger
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	SymbolInfoInteger
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	SymbolInfoInteger
SYMBOL_SWAP_LONG	Long swap value	SymbolInfoDouble
SYMBOL_SWAP_MODE	Swap calculation model	SymbolInfoInteger
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Swaps are charged in money, in client deposit currency	SymbolInfoInteger
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Swaps are charged in money in margin currency of the symbol	SymbolInfoInteger
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Swaps are charged in money in base currency of the symbol	SymbolInfoInteger
SYMBOL_SWAP_MODE_DISABLED	Swaps disabled (no swaps)	SymbolInfoInteger
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Swaps are charged as the specified annual interest from the instrument price at calculation of swap (standard bank year is 360 days)	SymbolInfoInteger
SYMBOL_SWAP_MODE_INTEREST_OPEN	Swaps are charged as the specified annual interest from the open price of position (standard bank year is 360 days)	SymbolInfoInteger
SYMBOL_SWAP_MODE_POINTS	Swaps are charged in points	SymbolInfoInteger
SYMBOL_SWAP_MODE_REOPEN_BID	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the current Bid price +/- specified number of points (parameters SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)	SymbolInfoInteger
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the close price +/- specified number of points (parameters	SymbolInfoInteger

	SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)	
SYMBOL_SWAP_ROLLOVER3DAYS	Day of week to charge 3 days swap rollover	SymbolInfoInteger
SYMBOL_SWAP_SHORT	Short swap value	SymbolInfoDouble
SYMBOL_TICKS_BOOKDEPTH	Maximal number of requests shown in Depth of Market . For symbols that have no queue of requests, the value is equal to zero.	SymbolInfoInteger
SYMBOL_TIME	Time of the last quote	SymbolInfoInteger
SYMBOL_TRADE_CALC_MODE	Contract price calculation mode	SymbolInfoInteger
SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	SymbolInfoDouble
SYMBOL_TRADE_EXECUTION_EXCHANGE	Exchange execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_MARKET	Market execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request	SymbolInfoInteger
SYMBOL_TRADE_EXEMODE	Deal execution mode	SymbolInfoInteger
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	SymbolInfoInteger
SYMBOL_TRADE_MODE	Order execution type	SymbolInfoInteger
SYMBOL_TRADE_MODE_CLOSE_ONLY	Allowed only position close operations	SymbolInfoInteger
SYMBOL_TRADE_MODE_DISABLED	Trade is disabled for the symbol	SymbolInfoInteger
SYMBOL_TRADE_MODE_FULL	No trade restrictions	SymbolInfoInteger
SYMBOL_TRADE_MODE_LONG_ONLY	Allowed only long positions	SymbolInfoInteger
SYMBOL_TRADE_MODE_SHORT_ONLY	Allowed only short positions	SymbolInfoInteger
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to place Stop orders	SymbolInfoInteger

SYMBOL_TRADE_TICK_SIZE	Minimal price change	SymbolInfoDouble
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK_VALUE_PROFIT	SymbolInfoDouble
SYMBOL_TRADE_TICK_VALUE_LOSS	Calculated tick price for a losing position	SymbolInfoDouble
SYMBOL_TRADE_TICK_VALUE_PROFIT	Calculated tick price for a profitable position	SymbolInfoDouble
SYMBOL_VOLUME	Volume of the last deal	SymbolInfoInteger
SYMBOL_VOLUME_LIMIT	Maximum allowed aggregate volume of an open position and pending orders in one direction (buy or sell) for the symbol. For example, with the limitation of 5 lots, you can have an open buy position with the volume of 5 lots and place a pending order Sell Limit with the volume of 5 lots. But in this case you cannot place a Buy Limit pending order (since the total volume in one direction will exceed the limitation) or place Sell Limit with the volume more than 5 lots.	SymbolInfoDouble
SYMBOL_VOLUME_MAX	Maximal volume for a deal	SymbolInfoDouble
SYMBOL_VOLUME_MIN	Minimal volume for a deal	SymbolInfoDouble
SYMBOL_VOLUME_STEP	Minimal volume change step for deal execution	SymbolInfoDouble
SYMBOL_VOLUMEHIGH	Maximal day volume	SymbolInfoInteger
SYMBOL_VOLUMELOW	Minimal day volume	SymbolInfoInteger
TENKANSEN_LINE	Tenkan-sen line	Indicators Lines
TERMINAL_BUILD	The client terminal build number	TerminalInfoInteger
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	TerminalInfoInteger
TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on a computer	TerminalInfoString

TERMINAL_COMMUNITY_ACCO UNT	The flag indicates the presence of MQL5.community authorization data in the terminal	TerminalInfoInteger
TERMINAL_COMMUNITY_BALA NCE	Balance in MQL5.community	TerminalInfoDouble
TERMINAL_COMMUNITY_CONN ECTION	Connection to MQL5.community	TerminalInfoInteger
TERMINAL_COMPANY	Company name	TerminalInfoString
TERMINAL_CONNECTED	Connection to a trade server	TerminalInfoInteger
TERMINAL_CPU_CORES	The number of CPU cores in the system	TerminalInfoInteger
TERMINAL_DATA_PATH	Folder in which terminal data are stored	TerminalInfoString
TERMINAL_DISK_SPACE	Free disk space for the MQL5\Files folder of the terminal (agent), MB	TerminalInfoInteger
TERMINAL_DLLS_ALLOWED	Permission to use DLL	TerminalInfoInteger
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the terminal settings	TerminalInfoInteger
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	TerminalInfoInteger
TERMINAL_LANGUAGE	Language of the terminal	TerminalInfoString
TERMINAL_MAXBARS	The maximal bars count on the chart	TerminalInfoInteger
TERMINAL_MEMORY_AVAILABL E	Free memory of the terminal (agent) process, MB	TerminalInfoInteger
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, MB	TerminalInfoInteger
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal (agent), MB	TerminalInfoInteger
TERMINAL_MEMORY_USED	Memory used by the terminal (agent), MB	TerminalInfoInteger
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID	TerminalInfoInteger

	data for Push notifications	
TERMINAL_NAME	Terminal name	TerminalInfoString
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to smartphone	TerminalInfoInteger
TERMINAL_OPENCL_SUPPORT	The version of the supported OpenCL in the format of 0x00010002 = 1.2. "0" means that OpenCL is not supported	TerminalInfoInteger
TERMINAL_PATH	Folder from which the terminal is started	TerminalInfoString
TERMINAL_PING_LAST	The last known value of a ping to a trade server in microseconds. One second comprises of one million microseconds	TerminalInfoInteger
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI).	TerminalInfoInteger
TERMINAL_TRADE_ALLOWED	Permission to trade	TerminalInfoInteger
TERMINAL_X64	Indication of the "64-bit terminal"	TerminalInfoInteger
THURSDAY	Thursday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
TRADE_ACTION_DEAL	Place a trade order for an immediate execution with the specified parameters (market order)	MqlTradeRequest
TRADE_ACTION_MODIFY	Modify the parameters of the order placed previously	MqlTradeRequest
TRADE_ACTION_PENDING	Place a trade order for the execution under specified conditions (pending order)	MqlTradeRequest
TRADE_ACTION_REMOVE	Delete the pending order placed previously	MqlTradeRequest
TRADE_ACTION_SLTP	Modify Stop Loss and Take Profit values of an opened position	MqlTradeRequest
TRADE_RETCODE_CANCEL	Request canceled by trader	MqlTradeResult

TRADE_RETCODE_CLIENT_DISABLED_AT	Autotrading disabled by client terminal	MqlTradeResult
TRADE_RETCODE_CONNECTION	No connection with the trade server	MqlTradeResult
TRADE_RETCODE_DONE	Request completed	MqlTradeResult
TRADE_RETCODE_DONE_PARTIAL	Only part of the request was completed	MqlTradeResult
TRADE_RETCODE_ERROR	Request processing error	MqlTradeResult
TRADE_RETCODE_FROZEN	Order or position frozen	MqlTradeResult
TRADE_RETCODE_INVALID	Invalid request	MqlTradeResult
TRADE_RETCODE_INVALID_EXPIRATION	Invalid order expiration date in the request	MqlTradeResult
TRADE_RETCODE_INVALID_FILL	Invalid order filling type	MqlTradeResult
TRADE_RETCODE_INVALID_ORDER	Incorrect or prohibited order type	MqlTradeResult
TRADE_RETCODE_INVALID_PRICE	Invalid price in the request	MqlTradeResult
TRADE_RETCODE_INVALID_STOPS	Invalid stops in the request	MqlTradeResult
TRADE_RETCODE_INVALID_VOLUME	Invalid volume in the request	MqlTradeResult
TRADE_RETCODE_LIMIT_ORDERS	The number of pending orders has reached the limit	MqlTradeResult
TRADE_RETCODE_LIMIT_VOLUME	The volume of orders and positions for the symbol has reached the limit	MqlTradeResult
TRADE_RETCODE_LOCKED	Request locked for processing	MqlTradeResult
TRADE_RETCODE_MARKET_CLOSED	Market is closed	MqlTradeResult
TRADE_RETCODE_NO_CHANGES	No changes in request	MqlTradeResult
TRADE_RETCODE_NO_MONEY	There is not enough money to complete the request	MqlTradeResult
TRADE_RETCODE_ONLY_REAL	Operation is allowed only for live accounts	MqlTradeResult
TRADE_RETCODE_ORDER_CHANGED	Order state changed	MqlTradeResult

TRADE_RETCODE_PLACED	Order placed	MqlTradeResult
TRADE_RETCODE_POSITION_CLOSED	Position with the specified POSITION_IDENTIFIER has already been closed	MqlTradeResult
TRADE_RETCODE_PRICE_CHANGED	Prices changed	MqlTradeResult
TRADE_RETCODE_PRICE_OFF	There are no quotes to process the request	MqlTradeResult
TRADE_RETCODE_REJECT	Request rejected	MqlTradeResult
TRADE_RETCODE_REQUOTE	Requote	MqlTradeResult
TRADE_RETCODE_SERVER_DISABLED_AT	Autotrading disabled by server	MqlTradeResult
TRADE_RETCODE_TIMEOUT	Request canceled by timeout	MqlTradeResult
TRADE_RETCODE_TOO_MANY_REQUESTS	Too frequent requests	MqlTradeResult
TRADE_RETCODE_TRADE_DISABLED	Trade is disabled	MqlTradeResult
TRADE_TRANSACTION_DEAL_ADD	Adding a deal to the history. The action is performed as a result of an order execution or performing operations with an account balance.	MqlTradeTransaction
TRADE_TRANSACTION_DEAL_DELETE	Deleting a deal from the history. There may be cases when a previously executed deal is deleted from a server. For example, a deal has been deleted in an external trading system (exchange) where it was previously transferred by a broker.	MqlTradeTransaction
TRADE_TRANSACTION_DEAL_UPDATE	Updating a deal in the history. There may be cases when a previously executed deal is changed on a server. For example, a deal has been changed in an external trading system (exchange) where it was previously transferred by a broker.	MqlTradeTransaction
TRADE_TRANSACTION_HISTORY_ADD	Adding an order to the history as a result of execution or	MqlTradeTransaction

	cancellation.	
TRADE_TRANSACTION_HISTORY_DELETE	Deleting an order from the orders history. This type is provided for enhancing functionality on a trade server side.	MqlTradeTransaction
TRADE_TRANSACTION_HISTORY_UPDATE	Changing an order located in the orders history. This type is provided for enhancing functionality on a trade server side.	MqlTradeTransaction
TRADE_TRANSACTION_ORDER_ADD	Adding a new open order.	MqlTradeTransaction
TRADE_TRANSACTION_ORDER_DELETE	Removing an order from the list of the open ones. An order can be deleted from the open ones as a result of setting an appropriate request or execution (filling) and moving to the history.	MqlTradeTransaction
TRADE_TRANSACTION_ORDER_UPDATE	Updating an open order. The updates include not only evident changes from the client terminal or a trade server sides but also changes of an order state when setting it (for example, transition from ORDER_STATE_STARTED to ORDER_STATE_PLACED or from ORDER_STATE_PLACED to ORDER_STATE_PARTIAL , etc.).	MqlTradeTransaction
TRADE_TRANSACTION_POSITION	Changing a position not related to a deal execution. This type of transaction shows that a position has been changed on a trade server side. Position volume, open price, Stop Loss and Take Profit levels can be changed. Data on changes are submitted in MqlTradeTransaction structure via OnTradeTransaction handler. Position change (adding, changing or closing), as a result of a deal execution,	MqlTradeTransaction

	does not lead to the occurrence of TRADE_TRANSACTION_POSITION transaction.	
TRADE_TRANSACTION_REQUEST	Notification of the fact that a trade request has been processed by a server and processing result has been received. Only type field (trade transaction type) must be analyzed for such transactions in MqlTradeTransaction structure. The second and third parameters of OnTradeTransaction (request and result) must be analyzed for additional data.	MqlTradeTransaction
TUESDAY	Tuesday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
TYPE_BOOL	bool	MqlParam
TYPE_CHAR	char	MqlParam
TYPE_COLOR	color	MqlParam
TYPE_DATETIME	datetime	MqlParam
TYPE_DOUBLE	double	MqlParam
TYPE_FLOAT	float	MqlParam
TYPE_INT	int	MqlParam
TYPE_LONG	long	MqlParam
TYPE_SHORT	short	MqlParam
TYPE_STRING	string	MqlParam
TYPE_UCHAR	uchar	MqlParam
TYPE_UINT	uint	MqlParam
TYPE_ULONG	ulong	MqlParam
TYPE_USHORT	ushort	MqlParam
UCHAR_MAX	Maximal value, which can be represented by uchar type	Numerical Type Constants
UINT_MAX	Maximal value, which can be represented by uint type	Numerical Type Constants

ULONG_MAX	Maximal value, which can be represented by ulong type	Numerical Type Constants
UPPER_BAND	Upper limit	Indicators Lines
UPPER_HISTOGRAM	Upper histogram	Indicators Lines
UPPER_LINE	Upper line	Indicators Lines
USHORT_MAX	Maximal value, which can be represented by ushort type	Numerical Type Constants
VOLUME_REAL	Trade volume	Price Constants
VOLUME_TICK	Tick volume	Price Constants
WEDNESDAY	Wednesday	SymbolInfoInteger , SymbolInfoSessionQuote , SymbolInfoSessionTrade
WHOLE_ARRAY	Means the number of items remaining until the end of the array, i.e., the entire array will be processed	Other Constants
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	Other Constants