

# **RealTest User Guide**

# Table of Contents

<b>1. Welcome!</b>	<b>12</b>
<b>2. Computer Requirements</b>	<b>13</b>
<b>3. File Types</b>	<b>14</b>
<b>4. File Paths</b>	<b>15</b>
<b>5. Backups</b>	<b>16</b>
<b>6. Examples and Tutorials</b>	<b>17</b>
6.1. Tutorial 1 - SPY Crossover	21
6.2. Tutorial 2 - Simple Optimization	26
6.3. Tutorial 3 - Simple Scan	33
6.4. Tutorial 4 - ETF Rotation	36
<b>7. Software User Interface</b>	<b>42</b>
7.1. Common Menus	43
7.1.1. File Menu	43
7.1.2. Edit Menu	45
7.1.3. View Menu	47
7.1.4. Data Menu	48
7.1.5. Run Menu	49
7.1.6. Window Menu	50
7.1.7. Help Menu	50
7.2. Child Window Menus	51
7.2.1. Chart Menu	51
7.2.2. Graph Menu	52
7.2.3. List Menu	54
7.2.4. Log Menu	54
7.2.5. Plot Menu	54
7.2.6. ResGraph Menu	55
7.2.7. Results Menu	57
7.2.8. Scan Menu	58
7.2.9. Script Menu	59
7.2.10. Compare Menu	60
7.2.11. Trades Menu	61
7.3. Bars and Panels	62
7.3.1. Tool Bar	62
7.3.2. Status Bar	63
7.3.3. Settings Panel	64
7.3.4. Debug Panel	66
7.4. Text Editor Windows	69
7.4.1. Script Windows	69
7.4.2. Log Windows	70
7.5. List Windows	71
7.5.1. Results Windows	73
7.5.2. Trade List Windows	75
7.5.3. Scan Windows	78
7.5.4. Trade Comparison Windows	80
7.5.5. Other List Windows	82
7.6. Graphical Windows	82
7.6.1. Daily Stats Graphs	82
7.6.2. Optimization Results Graphs	86
7.6.3. Candlestick/Bar Charts	87
7.6.4. Trade Plots and Analysis	90
7.7. Program Options Dialog	100
7.8. Report Options Dialog	102
7.9. RealTest Function Key Reference	104
<b>8. Importing Bar Data</b>	<b>106</b>
8.1. Norgate Data Import	106
8.2. Yahoo Import	109

8.3. Tiingo Import.....	110
8.4. CSV Import.....	111
8.5. MetaStock Import.....	112
8.6. Multi-Source Import.....	113
8.7. The Symbol Information File.....	115
8.8. Futures Symbol Information.....	116
8.9. The Event List File.....	116
<b>9. Running Scans.....</b>	<b>119</b>
<b>10. Running Tests.....</b>	<b>120</b>
10.1. Multiple Tests and Optimization.....	120
10.2. Walk-Forward Tests.....	122
<b>11. Using an Imported Trade List.....</b>	<b>127</b>
<b>12. Using Command Line Mode.....</b>	<b>130</b>
<b>13. Using Multiple Instances.....</b>	<b>132</b>
<b>14. Analyzing Test Results.....</b>	<b>133</b>
14.1. Test Summary Report.....	134
14.2. Test Details Log.....	135
<b>15. Trading Your System.....</b>	<b>137</b>
15.1. Tomorrow's Orders.....	137
15.2. Generated Order Types.....	140
15.3. OrderClerk.....	142
15.4. CSV Order Baskets.....	142
15.5. IB Rebalance Tool.....	145
15.6. Alera Signal Files.....	146
15.7. Daily Setups Scan.....	147
15.8. Multi-Row Scan.....	149
15.9. Test Output Scan.....	150
<b>16. Backtest Engine Details.....</b>	<b>153</b>
16.1. Asset Allocation and Position Sizing.....	155
16.2. Capacity Constraints.....	156
16.3. Compounding.....	157
16.4. Split Handling.....	160
16.5. Dividend Handling.....	162
16.6. Intraday Fills With Daily Bars.....	166
16.7. Intraday Fill Sequence Assumptions.....	167
16.8. Bar Sizes and Multiple Timeframes.....	167
16.9. Calculation of Trade Excursions.....	168
16.10. The Current Bar in Formula Evaluation.....	169
16.11. Specifying a Time Stop.....	169
16.12. Number of Bars Required for Functions and Indicators.....	170
16.13. Scaling In or Out of Positions.....	172
16.14. Referring to Past Trades in Strategy Formulas.....	175
16.15. Testing Multi-Currency Strategies.....	175
<b>17. Realtest Script Language.....</b>	<b>179</b>
17.1. Bar Offsets.....	180
17.2. Date Constants.....	181
17.3. Other Constants.....	181
17.4. Symbol Constants.....	182
17.5. String Values.....	182
17.6. Script Comments.....	183
17.7. File Path Syntax.....	183
17.8. Output Format Specification.....	184
17.9. Script Sections.....	185
17.9.1. Import Section.....	185
17.9.2. Data Section.....	186
17.9.2.1. One-Pass Data Formulas.....	187
17.9.2.2. Self-Referential Items.....	189
17.9.2.3. Bar-Size-Specific Items.....	190
17.9.3. TestData Section.....	191

17.9.4. Scan and TestScan Sections .....	193
17.9.5. Settings Sections .....	194
17.9.6. Library Section.....	196
17.9.7. Strategy Section .....	197
17.9.7.1. Strategy Names .....	198
17.9.7.2. Special Strategy Types .....	198
17.9.7.3. Strategy Elements.....	199
17.9.7.4. Strategy Element Value Types and Defaults .....	200
17.9.8. Parameters Section .....	202
17.9.9. Results Section.....	203
17.9.10. Graphs Section .....	204
17.9.11. Trades Section.....	206
17.9.12. Charts Section .....	207
17.9.13. Include Section.....	208
17.10. Formula Syntax.....	209
17.10.1. Operators.....	210
17.10.2. Formula Evaluation.....	211
17.10.3. Breadth Tags / Cross-Sectional Functions .....	212
17.10.4. External Symbol or Strategy Reference .....	213
17.10.5. Special Syntax for Individual Futures Contract Testing.....	214
17.10.6. Statistics Values in Formulas .....	216
17.11. Syntax Element Categories.....	217
17.11.1. Script Sections.....	217
17.11.2. Settings .....	218
17.11.3. Import Specification .....	219
17.11.4. Strategy Elements .....	220
17.11.5. Bar Data Values .....	222
17.11.6. Indicator Functions .....	223
17.11.7. Multi-Bar Functions .....	224
17.11.8. Cross-Sectional Functions .....	225
17.11.9. General-Purpose Functions.....	226
17.11.10. String Functions.....	227
17.11.11. Stock/Contract Information .....	227
17.11.12. Current Position Information.....	228
17.11.13. Test Statistics Arrays .....	229
17.11.14. Trade Record Values.....	230
17.11.15. Trade Statistics Functions.....	231
17.12. Syntax Element Details .....	232
17.12.1. #Avg .....	232
17.12.2. #ByCII.....	232
17.12.3. #ByEcon .....	233
17.12.4. #ByGroup .....	233
17.12.5. #ByIndu .....	233
17.12.6. #ByListNum .....	234
17.12.7. #ByMkt.....	234
17.12.8. #BySect.....	235
17.12.9. #Count.....	235
17.12.10. #DenseRank.....	235
17.12.11. #Highest.....	236
17.12.12. #Lowest.....	236
17.12.13. #Median .....	237
17.12.14. #OnePerDate.....	237
17.12.15. #OnePerSym .....	237
17.12.16. #PercentRank.....	238
17.12.17. #Rank.....	238
17.12.18. #StdDev .....	239
17.12.19. #Sum .....	239
17.12.20. ?CII .....	239
17.12.21. ?Currency.....	239

17.12.22. ?Domicile .....	240
17.12.23. ?EconSect.....	240
17.12.24. ?EquityType.....	240
17.12.25. ?Exchange.....	240
17.12.26. ?IndGroup.....	241
17.12.27. ?Industry.....	241
17.12.28. ?ListingType .....	241
17.12.29. ?LocalTime .....	241
17.12.30. ?Name .....	242
17.12.31. ?ReportingCurrency .....	242
17.12.32. ?RunMode.....	242
17.12.33. ?Sector .....	242
17.12.34. ?Strategy.....	243
17.12.35. ?Symbol .....	243
17.12.36. ?Type .....	243
17.12.37. Abs.....	244
17.12.38. AccountSize.....	244
17.12.39. Adjustment.....	244
17.12.40. ADX.....	245
17.12.41. AEMA.....	245
17.12.42. AESD .....	246
17.12.43. Allocation .....	246
17.12.44. Ambiguity.....	247
17.12.45. Arg1-Arg9.....	248
17.12.46. ATR .....	248
17.12.47. BarDate.....	248
17.12.48. BarNum.....	249
17.12.49. BarsHeld .....	249
17.12.50. BarSize.....	249
17.12.51. BarsLeft .....	250
17.12.52. BarStart .....	250
17.12.53. BBBOT .....	250
17.12.54. BBBotF.....	251
17.12.55. BBPCT.....	251
17.12.56. BBPctF .....	252
17.12.57. BBTOP .....	252
17.12.58. BBTotF.....	253
17.12.59. BBTREND .....	253
17.12.60. BBTrendF .....	253
17.12.61. BBWIDTH.....	254
17.12.62. BBWidthF .....	254
17.12.63. Benchmark .....	255
17.12.64. Bound.....	255
17.12.65. CalendarSym.....	255
17.12.66. CashInOut.....	256
17.12.67. CashIntPct.....	257
17.12.68. CashList .....	257
17.12.69. Category (reference).....	258
17.12.70. Category (definition).....	258
17.12.71. CCI .....	259
17.12.72. Charts.....	259
17.12.73. CIIFamily .....	259
17.12.74. CIILevel .....	260
17.12.75. Classification.....	260
17.12.76. Close or C.....	261
17.12.77. CloseSlip .....	261
17.12.78. Combined (function).....	261
17.12.79. Combined (section) .....	262
17.12.80. Commission.....	262

17.12.81. Compounded .....	262
17.12.82. Constituency.....	263
17.12.83. Correl.....	264
17.12.84. Cosine.....	265
17.12.85. Cross .....	265
17.12.86. CountTrue .....	266
17.12.87. CRSI .....	266
17.12.88. CSVDateFmt .....	267
17.12.89. CSVDelim .....	267
17.12.90. CSVFields .....	267
17.12.91. CSVFile .....	268
17.12.92. CSVNumFmt .....	268
17.12.93. Currency.....	269
17.12.94. Data .....	269
17.12.95. DataFile .....	269
17.12.96. DataPath.....	270
17.12.97. DataSource.....	270
17.12.98. DataType .....	270
17.12.99. Date .....	271
17.12.100. DateBars .....	271
17.12.101. DateDay .....	272
17.12.102. DateMonth.....	272
17.12.103. DateYear .....	272
17.12.104. Day.....	272
17.12.105. DayOfWeek.....	273
17.12.106. DayOfYear .....	273
17.12.107. Days .....	274
17.12.108. DebugEntry.....	274
17.12.109. DebugExit.....	274
17.12.110. DebugTargetStop .....	275
17.12.111. Dividend .....	275
17.12.112. DIIDataCalc.....	276
17.12.113. EMA or XAvg .....	276
17.12.114. EndDate (Import).....	277
17.12.115. EndDate (Setting).....	277
17.12.116. EndOfMonth.....	277
17.12.117. EndOfWeek.....	278
17.12.118. EntryDate .....	278
17.12.119. EntryLimit.....	278
17.12.120. EntryRank.....	279
17.12.121. EntryScore .....	279
17.12.122. EntrySetup.....	280
17.12.123. EntrySkip.....	281
17.12.124. EntryStop .....	281
17.12.125. EntryTime.....	282
17.12.126. EntryTradeValue.....	283
17.12.127. ESD .....	283
17.12.128. Event .....	284
17.12.129. EventListFile .....	284
17.12.130. ExchangeMap.....	284
17.12.131. ExcludeIf .....	286
17.12.132. ExcludeList .....	286
17.12.133. ExitLimit .....	286
17.12.134. ExitLimitQty .....	287
17.12.135. ExitLimitTime .....	287
17.12.136. ExitNum .....	288
17.12.137. ExitQty .....	288
17.12.138. ExitRank.....	289
17.12.139. ExitRule.....	289

17.12.140. ExitScore .....	290
17.12.141. ExitStop .....	291
17.12.142. ExitStopQty .....	291
17.12.143. ExitStopTime .....	291
17.12.144. ExitTime .....	292
17.12.145. ExitTradeValue .....	293
17.12.146. Exp .....	293
17.12.147. Extern .....	293
17.12.148. Extra .....	294
17.12.149. F.xxx / F.xxx.Date.....	294
17.12.150. FillFraction.....	295
17.12.151. FillPrice .....	296
17.12.152. FillPriceAvg .....	296
17.12.153. FillPriceMax.....	296
17.12.154. FillPriceMin .....	297
17.12.155. FillQty .....	297
17.12.156. FillValue .....	297
17.12.157. FilterNum .....	298
17.12.158. Format .....	298
17.12.159. Fundamentals .....	299
17.12.160. FunBar .....	299
17.12.161. Graphs .....	300
17.12.162. High or H .....	300
17.12.163. Highest or HHV .....	301
17.12.164. HMA or HAvG .....	301
17.12.165. HolidayList.....	301
17.12.166. HVOL .....	302
17.12.167. IF .....	302
17.12.168. Import .....	303
17.12.169. Include.....	303
17.12.170. IncludeList.....	304
17.12.171. InXXX .....	305
17.12.172. InfoID .....	305
17.12.173. InfoDelist / InfoExpiry.....	305
17.12.174. InfoFloat .....	306
17.12.175. InfoGICS .....	306
17.12.176. InfoMargin.....	306
17.12.177. InfoShares.....	306
17.12.178. InfoTRBC.....	307
17.12.179. InList .....	307
17.12.180. IsExit .....	308
17.12.181. IsNan .....	308
17.12.182. IsOrder .....	310
17.12.183. IsSetup .....	310
17.12.184. Item .....	310
17.12.185. KAMA .....	311
17.12.186. KBBOT .....	312
17.12.187. KBTOP .....	312
17.12.188. KeepAdjusted.....	313
17.12.189. KeepRedundant.....	313
17.12.190. KeepTrades.....	314
17.12.191. Kurtosis.....	314
17.12.192. Left.....	315
17.12.193. LegacyMode.....	315
17.12.194. Length .....	315
17.12.195. Library.....	316
17.12.196. LimitExtra .....	316
17.12.197. LimitSlip .....	316
17.12.198. LinReg.....	317

17.12.199. ListNum.....	317
17.12.200. Log.....	318
17.12.201. LogFile.....	318
17.12.202. Low or L.....	319
17.12.203. Lowest or LLV.....	320
17.12.204. MA or Avg.....	320
17.12.205. MACD.....	320
17.12.206. MACDH.....	321
17.12.207. MACDS.....	321
17.12.208. MarginIntPct.....	322
17.12.209. MarkToMarket.....	322
17.12.210. Match.....	323
17.12.211. Max.....	324
17.12.212. MaxN.....	324
17.12.213. MaxEntries.....	324
17.12.214. MaxExposure.....	325
17.12.215. MaxInvested.....	326
17.12.216. MaxNewExp.....	326
17.12.217. MaxNewInv.....	327
17.12.218. MaxNewPos.....	327
17.12.219. MaxPerTurn.....	327
17.12.220. MaxPositions.....	328
17.12.221. MaxSameCat.....	328
17.12.222. MaxSameSym.....	329
17.12.223. MaxSetups.....	330
17.12.224. MDI.....	330
17.12.225. Median.....	330
17.12.226. Mid.....	331
17.12.227. Min.....	331
17.12.228. MinN.....	331
17.12.229. Month.....	332
17.12.230. NextOpen.....	332
17.12.231. NoNan.....	332
17.12.232. Notes.....	333
17.12.233. NoWeekends.....	333
17.12.234. NumBars.....	334
17.12.235. OBV.....	334
17.12.236. Open or O.....	334
17.12.237. OpenSlip.....	335
17.12.238. OrderClerkFolder.....	335
17.12.239. OrderMktAsLmtPct.....	336
17.12.240. OrderNote / OrderExtra.....	336
17.12.241. OrderPrice.....	338
17.12.242. OrderRank.....	338
17.12.243. OrderSettings.....	338
17.12.244. OrdersFile.....	339
17.12.245. OrdersInclude.....	339
17.12.246. OrdersLiveData.....	339
17.12.247. OrdersMode.....	340
17.12.248. OrdersNetLiq.....	340
17.12.249. OrdersTemplate.....	341
17.12.250. OrderSum.....	342
17.12.251. PadAlignSym.....	342
17.12.252. Padding.....	342
17.12.253. Parameters.....	343
17.12.254. PDI.....	343
17.12.255. Peak.....	344
17.12.256. PeakBars.....	344
17.12.257. PercentRank.....	345

17.12.258. PercentRankN .....	346
17.12.259. PointValue .....	346
17.12.260. PrevExitLimit .....	346
17.12.261. PrevExitStop .....	347
17.12.262. Product .....	347
17.12.263. PositionSum.....	348
17.12.264. PriceRound .....	348
17.12.265. QtyFinal .....	349
17.12.266. QtyPrice .....	349
17.12.267. QtyRound .....	350
17.12.268. QtyType .....	350
17.12.269. Quantity .....	351
17.12.270. Random .....	352
17.12.271. RandomSeed.....	352
17.12.272. Range or R .....	353
17.12.273. Rank .....	353
17.12.274. RankN .....	353
17.12.275. Reduce.....	354
17.12.276. Replace .....	354
17.12.277. Results.....	354
17.12.278. ResultsFile .....	355
17.12.279. Right.....	355
17.12.280. RiskFreeRateSym.....	355
17.12.281. ROC or PctChg .....	356
17.12.282. Round .....	356
17.12.283. RSI .....	357
17.12.284. RRSI .....	357
17.12.285. RsiF .....	358
17.12.286. S.Alloc.....	358
17.12.287. S.BPx .....	359
17.12.288. S.CashInOut .....	359
17.12.289. S.Comms.....	360
17.12.290. S.Compounded.....	360
17.12.291. S.Date.....	360
17.12.292. S.DDBars.....	360
17.12.293. S.DDDlr.....	360
17.12.294. S.DDPct.....	361
17.12.295. S.Dividends.....	361
17.12.296. S.Entries.....	361
17.12.297. S.EntryOrders.....	362
17.12.298. S.Equity.....	362
17.12.299. S.Exits.....	362
17.12.300. S.Exposure .....	363
17.12.301. S.First .....	363
17.12.302. S.Interest .....	363
17.12.303. S.Invested .....	364
17.12.304. S.LossBars .....	364
17.12.305. S.LossDlr .....	364
17.12.306. S.Losses .....	364
17.12.307. S.LossPct.....	365
17.12.308. S.LossPctAlloc .....	365
17.12.309. S.M2M.....	365
17.12.310. S.MAE .....	365
17.12.311. S.MFE .....	366
17.12.312. S.MaxAlloc.....	366
17.12.313. S.MaxDDBars .....	366
17.12.314. S.MaxDDDLr .....	366
17.12.315. S.MaxDDPct.....	367
17.12.316. S.MaxEquity.....	367

17.12.317. S.MinAlloc.....	367
17.12.318. S.MinEquity.....	367
17.12.319. S.NetDir .....	367
17.12.320. S.NetFx .....	368
17.12.321. S.NetPct .....	368
17.12.322. S.Number .....	368
17.12.323. S.Positions.....	369
17.12.324. S.RiskFreeRate.....	369
17.12.325. S.Setups.....	369
17.12.326. S.Slips.....	370
17.12.327. S.StartEquity.....	370
17.12.328. S.Stops .....	370
17.12.329. S.Targets.....	370
17.12.330. S.TradeBars .....	370
17.12.331. S.TradeDir .....	371
17.12.332. S.TradePct.....	371
17.12.333. S.TradePctAlloc .....	371
17.12.334. S.TWEQ .....	371
17.12.335. S.Usage.....	372
17.12.336. S.WinBars.....	372
17.12.337. S.WinDir.....	372
17.12.338. S.WinPct.....	372
17.12.339. S.WinPctAlloc.....	373
17.12.340. S.Wins .....	373
17.12.341. SAR .....	373
17.12.342. SarF.....	373
17.12.343. SaveAs .....	374
17.12.344. SaveChartsTo.....	374
17.12.345. SavePositionsAs .....	375
17.12.346. SaveScanAs .....	375
17.12.347. SaveStatsAs.....	375
17.12.348. SaveTestListAs .....	376
17.12.349. SaveTradesAs.....	376
17.12.350. SaveTradesType.....	376
17.12.351. Scan .....	377
17.12.352. ScanInclude .....	377
17.12.353. ScanNoDefCols.....	377
17.12.354. ScanNoHeader .....	378
17.12.355. ScanNoWindow .....	378
17.12.356. ScanSettings .....	378
17.12.357. Select.....	378
17.12.358. Sequence.....	379
17.12.359. Settings.....	380
17.12.360. SetupRank.....	380
17.12.361. SetupScore .....	380
17.12.362. SetupSkip .....	381
17.12.363. SetupSum.....	381
17.12.364. Shares or Contracts .....	381
17.12.365. Side (position) .....	382
17.12.366. Side (strategy) .....	382
17.12.367. Sign.....	382
17.12.368. SinceHigh .....	383
17.12.369. SinceLow .....	383
17.12.370. SinceTrue .....	384
17.12.371. Sine.....	384
17.12.372. Skewness .....	385
17.12.373. SkipTestIf.....	385
17.12.374. Slippage .....	386
17.12.375. Slope .....	386

17.12.376. Spearman .....	387
17.12.377. Split .....	387
17.12.378. Sqr .....	388
17.12.379. SS .....	388
17.12.380. StartDate .....	388
17.12.381. StartDate .....	389
17.12.382. StartPercent .....	389
17.12.383. StatsGroup .....	389
17.12.384. StdDev .....	390
17.12.385. StdErr .....	390
17.12.386. STOC .....	391
17.12.387. StatsIncludeCash .....	392
17.12.388. Strategy .....	392
17.12.389. StrategyScore .....	393
17.12.390. StratNum .....	393
17.12.391. StopSlip .....	393
17.12.392. Sum .....	394
17.12.393. SumSince .....	394
17.12.394. SumSQ .....	395
17.12.395. Symbol .....	395
17.12.396. SymChangeList .....	396
17.12.397. SymInfoFile .....	397
17.12.398. SymNum .....	397
17.12.399. T.Bars .....	398
17.12.400. T.CommIn .....	398
17.12.401. T.CommOut .....	399
17.12.402. T.DateIn .....	399
17.12.403. T.DateOut .....	399
17.12.404. T.Div .....	399
17.12.405. T.Fraction .....	399
17.12.406. T.FxIn .....	400
17.12.407. T.FxOut .....	400
17.12.408. T.Highest .....	400
17.12.409. T.Lowest .....	400
17.12.410. T.NetFx .....	401
17.12.411. T.NetPct .....	401
17.12.412. T.Points .....	401
17.12.413. T.PriceIn .....	401
17.12.414. T.PriceOut .....	402
17.12.415. T.Profit .....	402
17.12.416. T.PtVal .....	402
17.12.417. T.QtyIn .....	402
17.12.418. T.QtyOut .....	402
17.12.419. T.Reason .....	403
17.12.420. T.Side .....	403
17.12.421. T.SlipIn .....	404
17.12.422. T.SlipOut .....	404
17.12.423. T.SplitIn .....	404
17.12.424. T.SplitOut .....	404
17.12.425. T.Strat .....	404
17.12.426. T.TimeIn .....	405
17.12.427. T.TimeOut .....	405
17.12.428. T.ValueIn .....	405
17.12.429. T.ValueOut .....	405
17.12.430. Tangent .....	406
17.12.431. Template .....	406
17.12.432. TargetPrice .....	406
17.12.433. TestInclude .....	407
17.12.434. TestData .....	407

17.12.435. TestName.....	407
17.12.436. TestOutput.....	408
17.12.437. TestScan.....	408
17.12.438. TestScanAllDates.....	409
17.12.439. TestSettings.....	409
17.12.440. This.....	409
17.12.441. TickSize.....	410
17.12.442. TLAdjusted.....	410
17.12.443. TLDateFmt.....	411
17.12.444. TLFields.....	411
17.12.445. TLIgnoreRules.....	412
17.12.446. TLStratName.....	412
17.12.447. TLTimeShift.....	413
17.12.448. TLValueIn.....	413
17.12.449. TLValueOut.....	413
17.12.450. ToDate.....	414
17.12.451. ToLower.....	414
17.12.452. ToNum.....	415
17.12.453. ToUpper.....	415
17.12.454. Top.....	415
17.12.455. TradeList.....	416
17.12.456. Trades.....	416
17.12.457. TradeStatAvg.....	416
17.12.458. TradeStatMax.....	417
17.12.459. TradeStatMin.....	417
17.12.460. TradeStatStdDev.....	417
17.12.461. TradeStatSum.....	418
17.12.462. Trough.....	418
17.12.463. TroughBars.....	419
17.12.464. TrueInRow.....	419
17.12.465. TrueRange or TR.....	420
17.12.466. UntilTrue.....	420
17.12.467. Update.....	420
17.12.468. UseAvailableBars.....	421
17.12.469. Using.....	421
17.12.470. Volume or V.....	422
17.12.471. WalkForward.....	422
17.12.472. Week.....	422
17.12.473. WhenTrue.....	423
17.12.474. WMA or WAvg.....	423
17.12.475. Year.....	424
17.12.476. YInt.....	424

# 1. Welcome!

---



## Welcome to RealTest!

RealTest is a multi-strategy portfolio-level backtesting tool. Trading systems modeled in RealTest can use any number of different trading instruments and can include any number of different trading strategies. This ability to easily model multi-strategy multi-instrument systems opens the door to a diversified approach to systematic trading that is difficult or impossible to test with other software.

There is a lot of information in this User Guide. Start by diving in and trying the four brief **Tutorials** with the software. After that, go back and forth between study and experimentation. There is also a series of video tutorials available on the mhptrading **YouTube** channel.

While evaluating and using the software feel free to also join the **RealTest User Forum**.

As you peruse the contents panel on the left, please note that the expandable items are also topics with their own text. Click on the item name (e.g. **Software User Interface** or **Backtest Engine Details**), not just the > symbol, in order to view this information.

If you happen to find any typos or have any suggestions about how this document could be more helpful, please don't hesitate to communicate them to [help@mhptrading.com](mailto:help@mhptrading.com).

Thank you!

## 2. Computer Requirements

---

RealTest is written in the *C programming language* using the *native Windows API* (as opposed to MFC or .NET). It was originally a 32-bit application, then for some time had both 32-bit and 64-bit versions. Now only the 64-bit version is maintained.

In many ways, RealTest is a *vintage 1990s-style Windows desktop application*.

There are no external libraries or components required, so installation is simple and clean.

The only change the installer makes to the *Windows Registry* is to associate the **file types** .RTS, .RTR and .RTD with this application.

Persistent settings are stored in the **realtest.ini** file in the program's installation directory, rather than in the Registry. RealTest must therefore be installed to a directory which has write permission.

With no data loaded, RealTest occupies less than 10mb of RAM. **Data** in memory occupies 64 bytes for every bar of every stock, plus 8 additional bytes per bar for each user-created data item in the active script.

In practice, a set of strategies using a universe of all US common stocks going back 10 years can be tested on a computer with 4GB of RAM. For best results when including more than 10 years and/or delisted symbols, 16GB of RAM is recommended.

RealTest will use up to 32 CPU threads if available. Multi-threading is only used for importing data and calculating your custom Data column formulas. The backtest engine is single-threaded but very efficient. Any CPU made in the past decade or so will be fast enough to run lots of tests in a small amount of time.

Your screen should have at least 1920x1080 pixels. RealTest is "DPI-aware" and fully supports higher resolutions such as 4K and/or scaling factors other than 100%.

RealTest also works fine on a cloud-based virtual machine, or a Mac running BootCamp or Parallels.

# 3. File Types

---

RealTest registers three file types with Windows:

-  RealTest **Data** (.RTD)
-  RealTest **Script** (.RTS)
-  RealTest **Results** (.RTR)

 **Data files** are created by running scripts that include **Import** definitions.

Currently supported data sources are:

- **Norgate Data** (this is the recommended data source for use with RealTest and is fully integrated)
- **Yahoo Finance** (quick and easy free data source, but with some quality issues and limitations)
- **Tiingo** (offers free and paid options, both of which require registration to get an API key)
- local **CSV** files (comma-delimited text, one file per symbol, one row per daily bar -- if you have CSV data you can use it)
- local **MetaStock** databases

 **Script files** are created by writing **Scripts** using the RealTest script editor, or any external text file editor of your choice.

A script is a plain text file containing a collection of parameters and formulas, organized in sections and sub-sections that correspond to the various tasks involved in the trading system research workflow.

 **Results files** are created by running scripts that include **Strategy** definitions and then saving the **Results** window contents.

A Results window or RTR file contains records of one or more tests that were run.

Each of these test records includes:

- The script and parameters that were used to run the test
- Daily summary statistics for each day in the test
- List of trades from the test (optional, and optionally including skipped trades)

As with data files, this information is stored in the same binary format that it occupies in memory.

## 4. File Paths

---

The default installation folder for RealTest is *C:\RealTest*.

You have the option to specify a different path during installation if desired.

The installer creates the following sub-folders under this folder:

- **Scripts** - a place to put your own scripts
- **Scripts\Examples** - a set of example scripts
- **Data** - a place to put your imported data files
- **Output** - a place to organize all kinds of RealTest output files, such as scans, trade lists, results files, etc.
- **Output\Info** - stock information reports will go here
- **Output\Logs** - test log files will go here
- **Output\Orders** - generated order lists will go here
- **Output\Reports** - test summary reports will go here

The default parent folder of all of the above is the installation folder, e.g. *C:\RealTest*.

The **Program Options** dialog box (via the View menu) provides a way to specify different locations for any of the main three default paths (Scripts, Data, Output).

Please note that whenever you decide to change a default path location, you are responsible for renaming and/or moving the actual folders yourself.

## 5. Backups

---

RealTest creates a folder called "Backups" within the installation folder (by default *C:\RealTest\Backups*).

Every time a script file (\*.rts) is saved to disk, the previous version is automatically copied to the Backups folder.

If another file of the same name already exists in that folder, it is overwritten.

The same is done for the RealTest.ini file, which is where all your user interface settings are stored and remembered.

The purpose of this Backups folder is to provide a way to go "one version back" if ever needed, or to retrieve a script in case it was accidentally deleted or overwritten.

It is also, of course, highly recommended to add your RealTest scripts folder to your list of folders to be backed up by whatever automatic local or cloud backup services you are using.

At the same time, it is advisable to exclude data files (\*.rtd) from such automatic backups, as they can be quite large and are easy to recreate. (It is also advisable to exclude your *Data* folder and/or \*.rtd files from anti-virus scanning for better performance.)

## 6. Examples and Tutorials

---

The topics that follow provide four simple guided **Tutorials** that can be used when getting to know RealTest.

There is also a series of tutorial videos on the [mhptrading YouTube channel](#).

The **Examples** folder (C:\RealTest\Scripts\Examples) contains the following:

---

### **Tutorial Scripts**

- **sample1.rts** - simple moving average crossover strategy for SPY (see **Tutorial 1**)
  - **sample2.rts** - a parameterized version of sample1, allowing for optimization (see **Tutorial 2**)
  - **sample2a.rts** - a three-parameter version of sample2 (also used in **Tutorial 2**)
  - **sample\_scan.rts** - introduces the Data section and the scanner (see **Tutorial 3**)
  - **sector\_etfs.rts** - implements a monthly rotational strategy for the S&P 500 sector ETFs (see **Tutorial 4**)
- 

### **Import Examples**

- **import\_csv.rts** - shows how to specify CSV data import (data not provided)
  - **import\_ms.rts** - shows how to specify MetaStock data import (data not provided)
  - **import\_multi.rts** - shows how to combine data from multiple sources into a single RTD file
  - **import\_norgate.rts** - shows how to import data directly from **Norgate NDU**, and how to access Norgate's **index constituency data**
  - **import\_tiingo.rts** - shows how to import data from Tiingo, if you have an API key
  - **import\_yahoo.rts** - shows how to import all the SPX components from Yahoo (see **Tutorial 3**)
  - **djia\_earnings.rts** - shows how to use an **Event List File** to include earnings dates in your imported data
  - **djia\_make\_syminfo.rts** - shows how to create a **Symbol Information File** with Norgate metadata for use with other data sources
  - **djia\_use\_syminfo.rts** - shows how to use a symbol information file to add metadata to a Yahoo import
  - **actual\_trades.rts** - shows how to "test" a list of actual trades using an **Imported Trade List** file, includes templates for various trade list formats
- 

### **Scanning Examples**

- **breadth.rts** - shows how to use cross-sectional formulas in the **Data Section** to calculate and chart a market-breadth indicator
- **gics\_indu\_rank.rts** - shows how to find the top X stocks in the top Y industries using GICS classification
- **ibd\_rs.rts** - shows how to calculate IBD's relative strength ranks for any universe of stocks
- **index\_breadth.rts** - shows the best way to calculate breadth statistics using only historical index constituents each day
- **industry\_indices.rts** - shows how to use Norgate's "Corresponding Industry Index" features to know the industry relative strength of any stock
- **multi\_filter\_scan.rts** - shows how to write a scan that creates multiple rows per symbol per date with different data in each row
- **opex\_and\_vxex.rts** - calculates monthly options expiration and VIX futures expiration dates

- **sctr.rts** - shows how to calculate "StockCharts Trend Rank" for any universe of stocks
- **stockbee\_mm.rts** - shows how to calculate current and historical values for the **Stockbee Market Monitor** breadth indicators

---

### **Indicators and Techniques**

- **actual\_trades.rts** - shows how to "play back" a list of trades from a CSV file
- **anchored\_vwap.rts** - shows how to calculate volume-weighted average price anchored to a specific past date
- **annual\_taxes.rts** - calculates capital gains and dividend income taxes for any script by simply including it at the end
- **beta\_indicator.rts** - shows how to calculate "beta" of individual stocks vs. an index
- **dividend\_yield\_series.rts** - shows how to calculate the dividend yield series for all stocks
- **ehlers\_windows.rts** - implements a set of indicators presented by John Ehlers in the Sept. 2021 issue of TASC
- **fundamentals.rts** - shows how to import and scan for Norgate current fundamentals
- **management\_fees.rts** - calculates management and performance fees for any script by simply including it at the end
- **martingale.rts** - shows how to allow multiple positions in one symbol within a single strategy
- **oc\_all\_actual.rts** - shows how to structure a script to play back OrderClerkTrades.csv with multiple strategies (possibly from different scripts)
- **scale\_in.rts** - shows how to scale into a position as price moves in your favor, keeping risk smaller
- **supertrend.rts** - shows how to calculate and plot the "supertrend" indicator (dual trailing stops)
- **trail\_half.rts** - shows how to divide a strategy into two half-allocation strategies to implement "take half off at a target and trail the rest"
- **yield\_rank.rts** - shows how to calculate annualized dividend yield from actual dividends and rank stocks by their yields

---

### **Single-Strategy Systems**

- **cii\_rotate.rts** - shows how to use Norgate's *Corresponding Industry Index* capability to hold the top 3 stocks of the top 5 industries each month
- **clenow\_stocks\_on\_move.rts** - a simple implementation of this strategy from a well-known book
- **dalio\_all\_weather.rts** - implements the Ray Dalio "All-Weather Portfolio"
- **dalio\_all\_weather\_maxdiff.rts** - same as above but waits for X% difference before rebalancing
- **dividend\_capture.rts** - shows how to use Norgate dividend data and a slightly unusual approach to capturing dividends
- **flipper.rts** - a simple implementation of the Nick Radge "Flipper" strategy idea
- **higher\_lows.rts** - a trend-following strategy that uses a series of higher pivot lows as an entry signal
- **hybrid\_asset\_allocation.rts** - implements this **Keller and Keuning** monthly ETF rotational strategy
- **keller\_baa.rts** - implements the Keller **Bold Asset Allocation** monthly ETF rotational strategy
- **keltner\_pullback.rts** - a pullback-after-momentum strategy based on the work of Adam Grimes, uses risk-based position sizing
- **ndx\_rotate.rts** - a simple momentum-based rotational strategy using Nasdaq 100 component

stocks

- **ndx\_rotate\_factor\_test.rts** - shows how to loop through a set of completely different formulas using the optimizer
- **ndx\_rotate\_weekly\_reduce.rts** - a variation of ndx\_rotate that reduces position size during drawdowns
- **oex\_tf\_fresh\_signal.rts** - simple trend-following strategy that only enters on a "fresh signal"
- **radge\_bbo.rts** - the "BBO" strategy from *Holy Grails* by Nick Radge
- **sector\_etfs\_breadth.rts** - monthly rotation using sector constituent breadth to rank the ETFs
- **simple\_day\_trade.rts** - implements a simple long-only "day trading strategy" that enters with limit orders and exits market-on-close (MOC)
- **simple\_day\_trade\_basket\_orders.rts** - shows how easy it is to generate a daily *IB Basket Trader* order file for this strategy
- **simple\_day\_trade\_basket\_scan.rts** - shows how produce the same order list using a Scan, in case you ever needed to
- **spy\_tlt\_uis.rts** - demonstrates using **Walk-Forward** optimization to implement a SPY/TLT strategy from [an article](#)
- **tf\_dynamic\_size.rts** - a simple trend-following strategy that resizes positions when distance from stop changes
- **tf\_sell\_half\_at\_1R.rts** - a simple trend-following strategy that sells half at a 1R target and trails a stop for the remainder
- **trend\_following\_basics.rts** - illustrates a variety of stop techniques, risk-based position sizing, and "fresh breakout" detection
- **vigilant\_asset\_allocation.rts** - implements this **Keller and Keuning** monthly ETF rotational strategy
- **vxx\_long\_short.rts** - be long or short the VXX ETF depending on VIX term structure
- **weekly\_moc\_asx** - a simplified weekly mean-reversion strategy
- **weekly\_moc\_asx\_daily\_weekly.rts** - the above implemented as a weekly strategy within a daily script
- **weekly\_moc\_asx\_daily\_daily.rts** - the above implemented as weekly logic in a daily strategy
- **weekly\_trend\_following.rts** - a weekly Russell 1000 trend-following strategy similar to the Nick Radge "Weekend Trend Trader"

---

### **Multi-Strategy Systems**

- **bensdorp\_book.rts** - an implementation of the strategies described in *Automated Stock Trading Systems* by Laurens Bensdorp
- **combined.rts** - demonstrates how to combine strategies that use different symbol universes
- **combined\_multi\_bar\_size.rts** - shows how strategies in a script can each use their own bar size and refer to external bar sizes
- **combined\_rebalance.rts** - a version of combined.rts that models running each strategy in its own account with periodic transfers
- **goal\_30\_15.rts** - a response to a challenge to show a set of strategies with > 30% ROR and < 15% MaxDD (from 2014-2021)
- **goal\_30\_15\_asx.rts** - a version of the above tailored for the Australian market (\$XAO constituents)
- **goal\_30\_15\_tsx.rts** - a version of the above tailored for the Canadian market (\$SPTSXconstituents)
- **mhp\_classic.rts** - provided for those who might be curious about the kinds of strategies

Marsten traded in the early 2000s

- **multi\_moc\_top\_down.rts** - combines four variations of simple\_day\_trade.rts using top-down setup selection
- **oex\_tf\_top\_down.rts** - combines three OEX trend-following strategies using top-down setup selection
- **two\_accounts.rts** - compound two strategies together in one account with combined with a third strategy in a separate account

---

### **Mean Reversion Theme and Variations**

- **mr\_sample.rts** - presents a somewhat sophisticated long/short mean-reversion strategy pair using Norgate data (and an example of "tail risk" in January 2021)
- **mr\_sample\_benchmark.rts** - adds a benchmark strategy to the mr\_sample example
- **mr\_sample\_common.rts** - contains the common elements of the following three examples, each of which includes this script
- **mr\_sample\_debug.rts** - demonstrates use of strategy debugging formulas to look under the hood of a running test, and also demonstrates the usefulness of the Library script section
- **mr\_sample\_hedged.rts** - adds an index ETF hedge to the mr\_sample strategy pair
- **mr\_sample\_orders.rts** - shows how to configure this pair of strategies for OrderClerk order generation
- **mr\_sample\_orders\_alera.rts** - shows how to configure the strategy pair for daily Alera Portfolio Manager signal file generation
- **mr\_sample\_orders\_basket.rts** - shows how to configure the strategy pair for daily IB Basket Trader order file generation
- **mr\_sample\_scan.rts** - demonstrates a possible **Daily Setups Scan** that could be used with the mr\_sample strategy pair
- **mr\_sample\_test\_scan.rts** - shows how to run a test+scan to generate a set of daily orders
- **mr\_sample\_tracking.rts** - shows how to test a strategy that only takes trades when its own equity curve is above a moving average (avoided above mentioned "tail risk"...)
- **mr\_sample\_tracking\_alloc.rts** - shows how to test a strategy that reduces its allocation when its drawdown increases
- **mr\_sample\_tradelist.rts** - shows how to combine strategies from separate scripts by using tradelist generation and playback
- **mr\_sample\_long\_only.rts** and **mr\_sample\_short\_only.rts** are also provided for use with the above example

---

### **Futures Examples**

- **cl\_term\_structure.rts** - plots the 8 most recent CL contracts and scans for the 12 most recent ones
- **es\_compare.rts** - using Norgate futures data, tests buy and hold of two continuous contract series (adjusted and unadjusted) vs. rolling through each individual contract
- **futures\_calendar\_spread.rts** - shows two ways to construct calendar spreads using Norgate futures data
- **futures\_trend\_follow\_simple.rts** - simplest trend-following strategy using continuous back-adjusted contract data only
- **futures\_volume\_rank.rts** - trend-following strategy that uses #Rank #ByMkt to trade the most active individual contract from each market and roll as needed
- **gc\_kelly.rts** - demonstrates how to model Kelly Criterion position sizing using a simple trend-following strategy for gold futures

- **turtles.rts** - an approximation of the original "Turtles" futures system, as documented by Curtis Faith
- **vx\_futures.rts** - be long or short the front-month VX futures contract depending on the term structure
- **vx\_term\_structure.rts** - illustrates how to plot the entire VX futures term structure in the indicator pane of a \$VIX chart

---

### Files Used by Examples Scripts

- **actual\_trades.csv** - the list of trades used in the actual\_trades.rts example
- **chartist\_api\_template.csv** - order list template file for "The Chartist Smart API" (Nick Radge) output format
- **djia\_earnings.csv** - a sample list of earnings dates for the DJIA components since 1/1/2019 (not guaranteed to be accurate)
- **djia\_info.csv** - a detailed symbol information metadata file for the DJIA components
- **djia\_syms.txt** - symbols of the current DJIA components
- **holidays.au.txt** - example Australian market HolidayList file to use when generating orders
- **holidays.us.txt** - example US market HolidayList file to use when generating orders
- **ib\_basket\_template.csv** - order list template file for IB Basket Trader output format
- **spx\_syms.txt** - symbols of the current S&P 500 components

---

## 6.1. Tutorial 1 - SPY Crossover

To quickly learn the basics of using RealTest, let's start with a very simple trading system: a moving average crossover signal for one symbol (SPY).

When you first run RealTest, you will see that this example script has already been opened for you in a **Script Window**:

The screenshot shows a script window titled "Active Script - C:\RealTest\SCRIPTS\Examples\Sample1.rts". The script content is as follows:

```

Notes:   Simplest Example -- 50/200 crossover on SPY
         to run for the first time, first click on Import, then click on Test
         to run the test again, just click Test (or press F5)

Import:  DataSource:   Yahoo
         IncludeList:  SPY
         StartDate:    1/1/1992
         SaveAs:       sample1.rtd

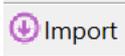
Settings: DataFile:    sample1.rtd
          StartDate:   Earliest
          EndDate:     Latest

Strategy: SPY_Crossover
          EntrySetup:  Avg(C,50) > Avg(C,200)
          ExitRule:    Avg(C,50) < Avg(C,200)
  
```

The script includes three sections:

- the **Import** section specifies how to obtain the data needed to run the backtest
- the **Settings** section tells RealTest what settings to use when running the test
- the **Strategy** section defines the trading strategy

To start, click on "Import" in the Tool Bar at the top of the main window.



If your computer is connected to the Internet, this will download the complete daily price and volume history for the SPY ETF from the Yahoo Finance website and then save it to your local disk in RealTest's binary data format as a file called sample1.rtd.

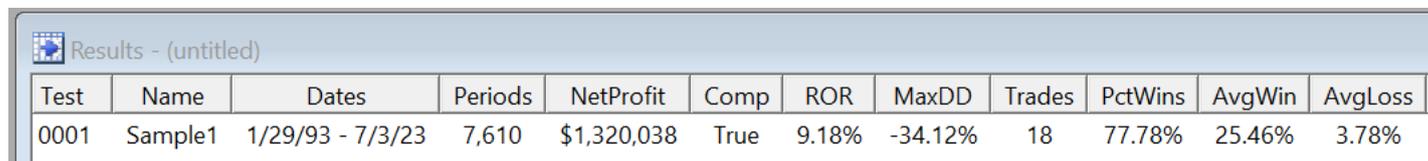
The import will probably take less than one second since there is only one symbol to retrieve.

Next, click on the "Test" button.



Since there's only one symbol, one strategy, and not many calculations required, this backtest will run nearly instantaneously, after which a new window will appear.

The **Results Window** shows the summary statistics for the test and provides access to underlying details:

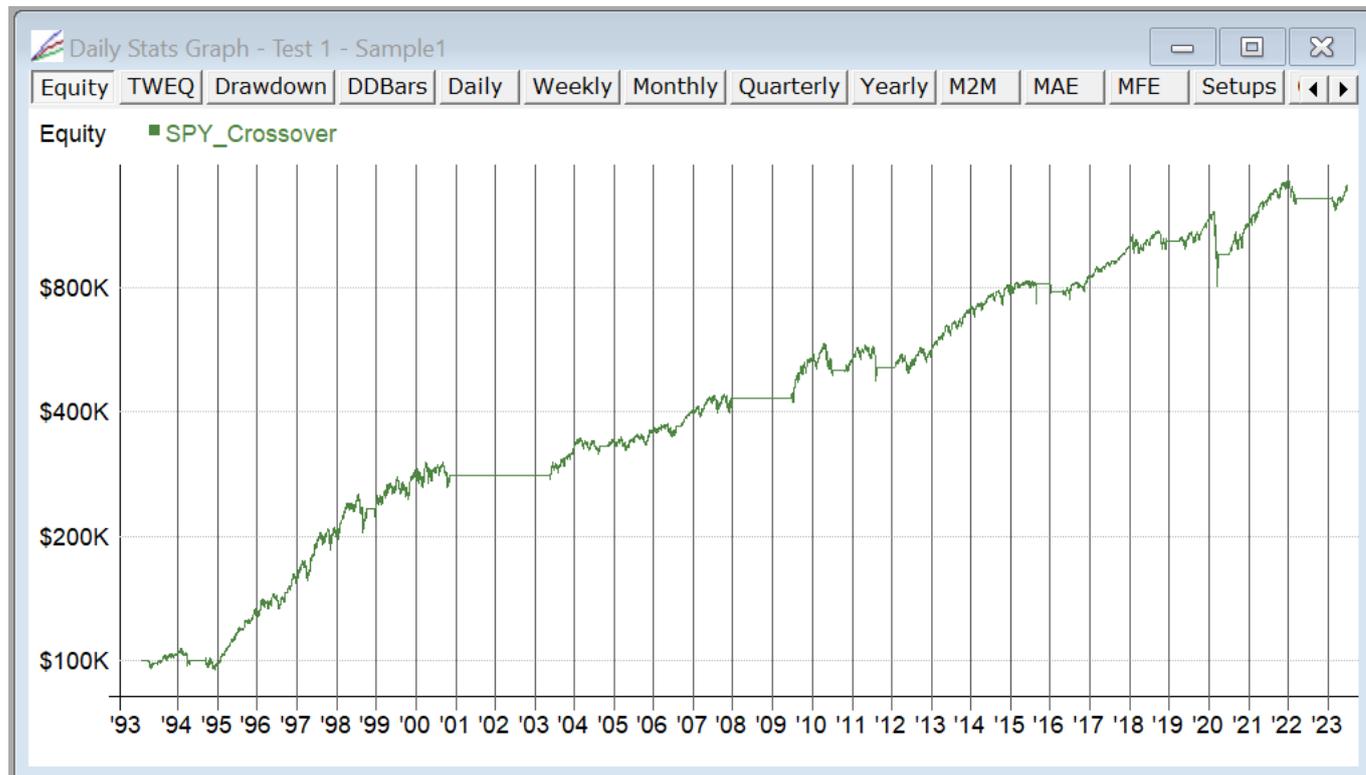
A screenshot of a window titled "Results - (untitled)". It contains a table with 12 columns: Test, Name, Dates, Periods, NetProfit, Comp, ROR, MaxDD, Trades, PctWins, AvgWin, and AvgLoss. The first row of data shows: 0001, Sample1, 1/29/93 - 7/3/23, 7,610, \$1,320,038, True, 9.18%, -34.12%, 18, 77.78%, 25.46%, 3.78%.

Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	Trades	PctWins	AvgWin	AvgLoss
0001	Sample1	1/29/93 - 7/3/23	7,610	\$1,320,038	True	9.18%	-34.12%	18	77.78%	25.46%	3.78%

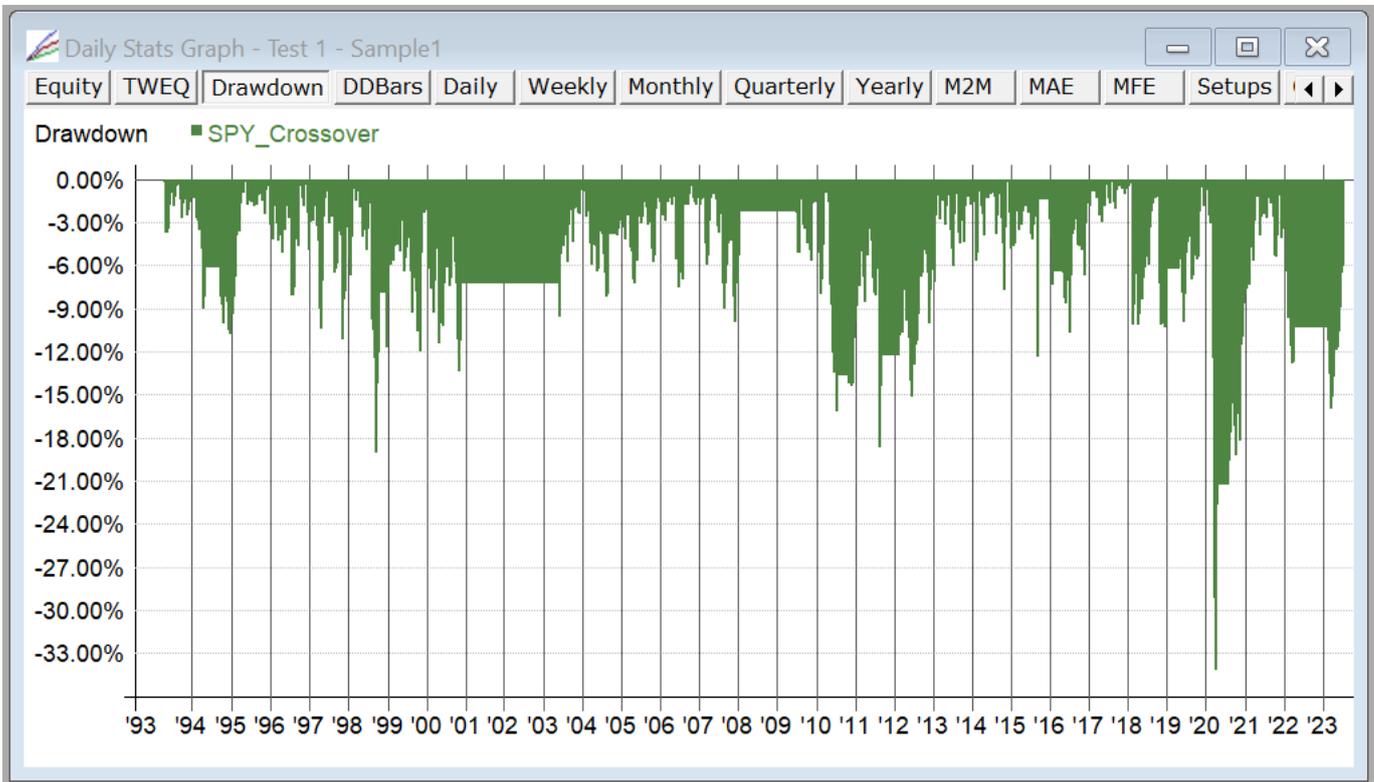
(There are other default columns, but they'd make the image too wide to be readable here.)

Double-click on the row of stats in the Results window to open the stats graphs.

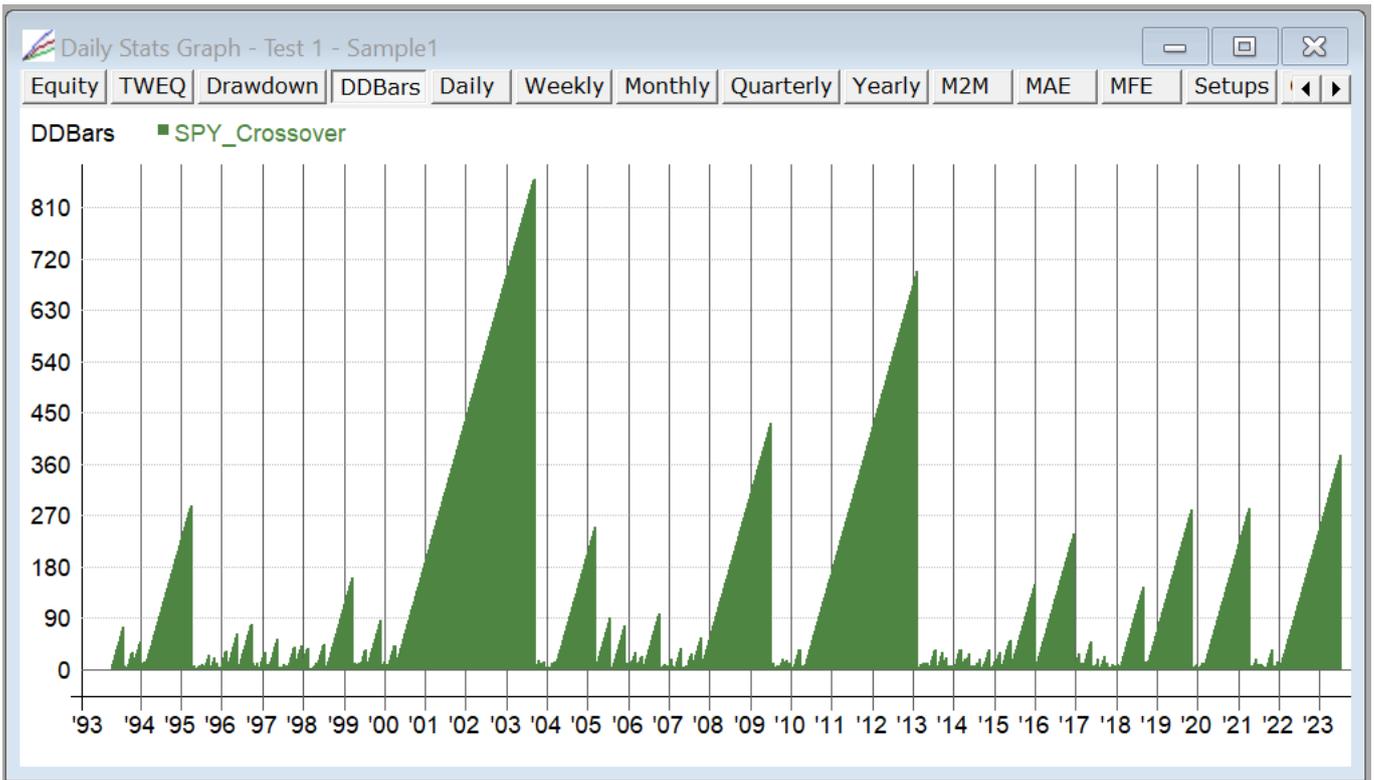
The **Daily Stats Graph** shows the equity curve:



Use the buttons along the top of this window to look at some of the other default stats graph types:



Above was %drawdown (not so great in March 2020), below is drawdown duration.

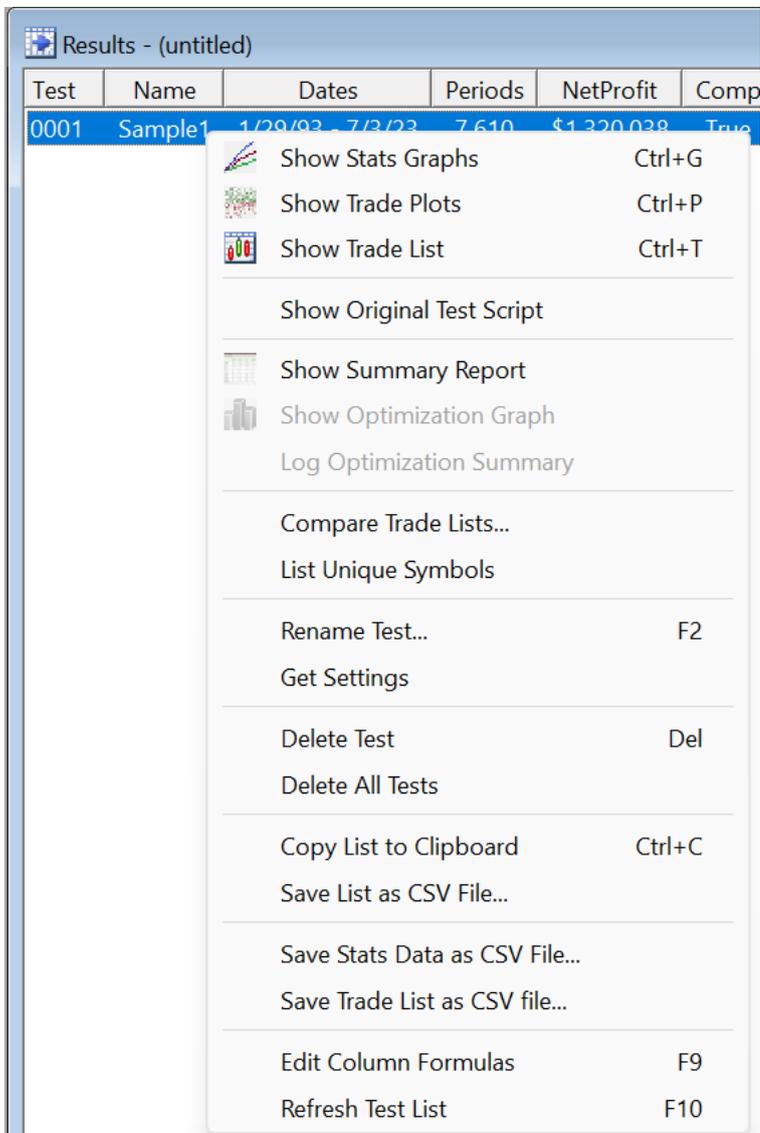


You can also use the left and right arrow keys to cycle through the different graph types.

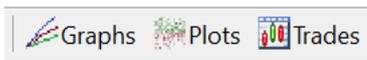
When there is more than one test in the results window, the up and down arrow keys let you easily see the same graph for different tests.

The contents of the **results window** columns and **graph window** items are specified by formula and completely customizable.

Drilling down to the next level, a right-click on a test result row reveals the following popup menu:



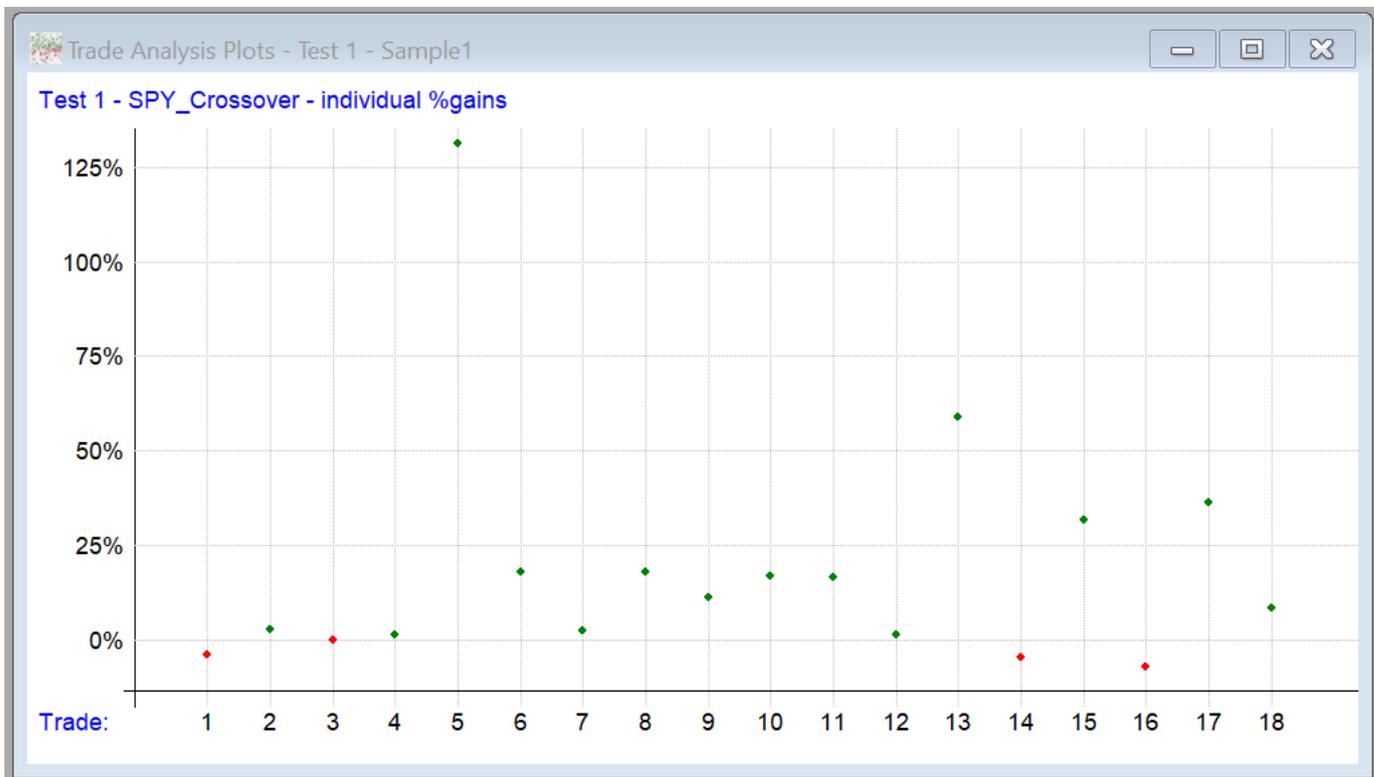
The first three items are the ones you'll use most often, so they also appear in the Tool Bar:



(Note that you must be in the Results window with a row of stats selected for these Tool Bar buttons to be active.)

Click on "Graphs" to open the stats graphs for a test (in this example you already did this by double-clicking on the stats row).

Click on "Plots" to access a variety of **graphical trade-level analysis** tools, such as a scatter plot:



Click on "Trades" to view the detailed **trade list**:

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn	DateOut	TimeOut	QtyOut	PriceOut	Reason	Bars	PctGain	Profit
00001	SPY_Crossover	SPY	Long	4/14/93	open	2,222	45.0313	4/27/93	open	2,222	43.3438	exit rule	9	-3.75%	(\$3,749.62)
00002	SPY_Crossover	SPY	Long	4/29/93	open	2,198	43.875	5/26/93	open	2,198	45.1563	exit rule	19	2.92%	\$2,816.30
00003	SPY_Crossover	SPY	Long	6/22/93	open	2,221	44.6563	6/23/93	open	2,221	44.625	exit rule	1	-0.07%	(\$69.52)
00004	SPY_Crossover	SPY	Long	7/6/93	open	2,215	44.625	4/20/94	open	2,215	44.4063	exit rule	201	1.47%	\$1,451.49
00005	SPY_Crossover	SPY	Long	9/15/94	open	2,135	47.1719	9/30/98	open	2,135	103.5	exit rule	1,021	131.46%	\$132,397.97
00006	SPY_Crossover	SPY	Long	12/9/98	open	1,966	118.6875	11/5/99	open	1,966	138.625	exit rule	229	18.05%	\$42,124.50
00007	SPY_Crossover	SPY	Long	11/12/99	open	1,985	139.25	10/31/00	open	1,985	141.0156	exit rule	244	2.30%	\$6,367.09
00008	SPY_Crossover	SPY	Long	5/16/03	open	2,958	94.89	8/19/04	open	2,958	109.81	exit rule	317	17.92%	\$50,300.79
00009	SPY_Crossover	SPY	Long	11/8/04	open	2,827	116.98	7/20/06	open	2,827	126.12	exit rule	427	11.35%	\$37,548.21
00010	SPY_Crossover	SPY	Long	9/12/06	open	2,830	130.56	12/24/07	open	2,830	148.82	exit rule	323	17.11%	\$63,202.39
00011	SPY_Crossover	SPY	Long	6/24/09	open	4,839	90.16	7/7/10	open	4,839	103.13	exit rule	260	16.72%	\$72,967.28
00012	SPY_Crossover	SPY	Long	10/25/10	open	4,270	119.14	8/15/11	open	4,270	119.19	exit rule	203	1.58%	\$8,044.68
00013	SPY_Crossover	SPY	Long	2/1/12	open	3,909	132.29	8/31/15	open	3,909	198.11	exit rule	900	59.02%	\$305,179.54
00014	SPY_Crossover	SPY	Long	12/18/15	open	3,995	202.77	1/12/16	open	3,995	193.82	exit rule	15	-4.41%	(\$35,755.25)
00015	SPY_Crossover	SPY	Long	4/26/16	open	3,752	209.04	12/10/18	open	3,752	263.37	exit rule	661	31.71%	\$248,708.82
00016	SPY_Crossover	SPY	Long	4/2/19	open	3,608	286.04	3/31/20	open	3,608	260.56	exit rule	251	-6.88%	(\$71,034.30)
00017	SPY_Crossover	SPY	Long	7/10/20	open	3,055	314.31	3/15/22	open	3,055	419.77	exit rule	423	36.30%	\$348,557.17
00018	SPY_Crossover	SPY	Long	2/3/23	open	3,140	411.59	7/3/23	close	3,140	443.79	end of test	102	8.59%	\$110,980.16

As with the results window, there are other columns to the right, and the content can be **customized**.

Double-click on any row in the trade list to view that trade on a **candlestick or bar chart**:



The chart automatically aligns with the trade timeframe. Press the UP or DOWN keys to cycle the chart through the trades in the list.

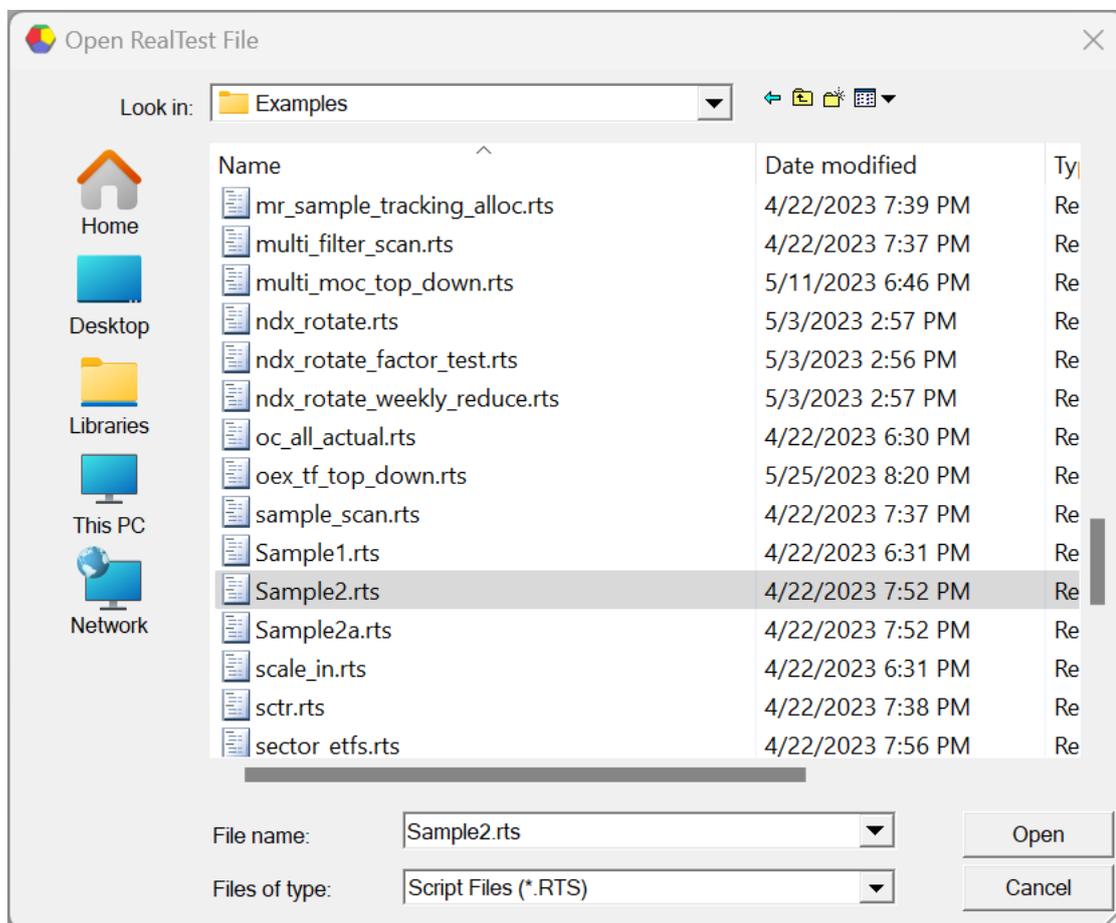
That's the end of this first tutorial!

To learn how to optimize the two parameters in this strategy, proceed to **Tutorial 2**.

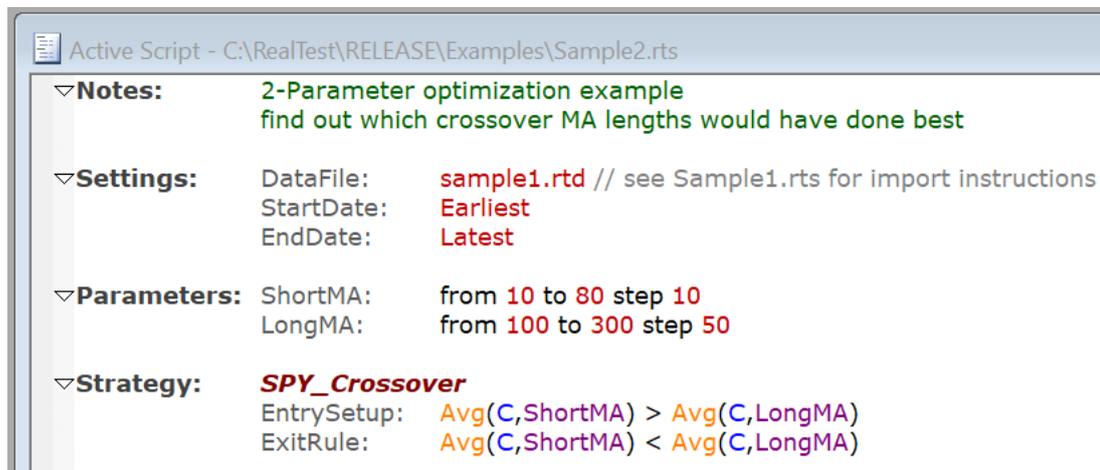
## 6.2. Tutorial 2 - Simple Optimization

To start the second tutorial, first close all the child windows that were opened for the first tutorial. (This is not required, it just makes it easier to keep track of what you're doing.)

Click the File / Open icon,  then navigate to the Examples folder in the RealTest installation directory.



Select "Sample2.rts" and open it.



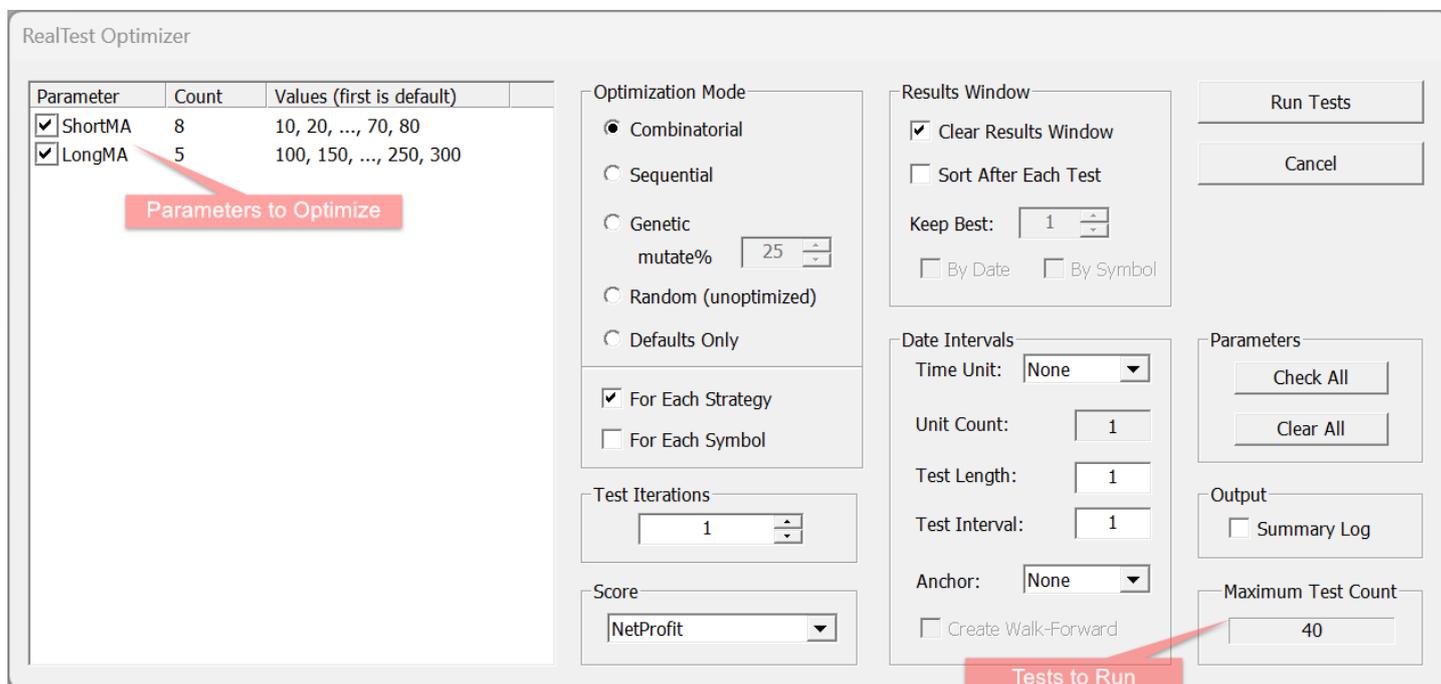
This script will use the same data file as the one in the first tutorial, so there is no Import section.

Rather than running a single test, we will now try the **RealTest Optimizer**.

Notice the **Parameters section** in the script above. This defines named parameters that can be referred to in any strategy formula.

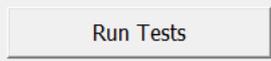
Rather than hard-coding the 50/200 moving average crossover strategy as in Sample1.rts, the two moving average lengths are now parameters.

Press  **Optimize** and the following dialog will appear:

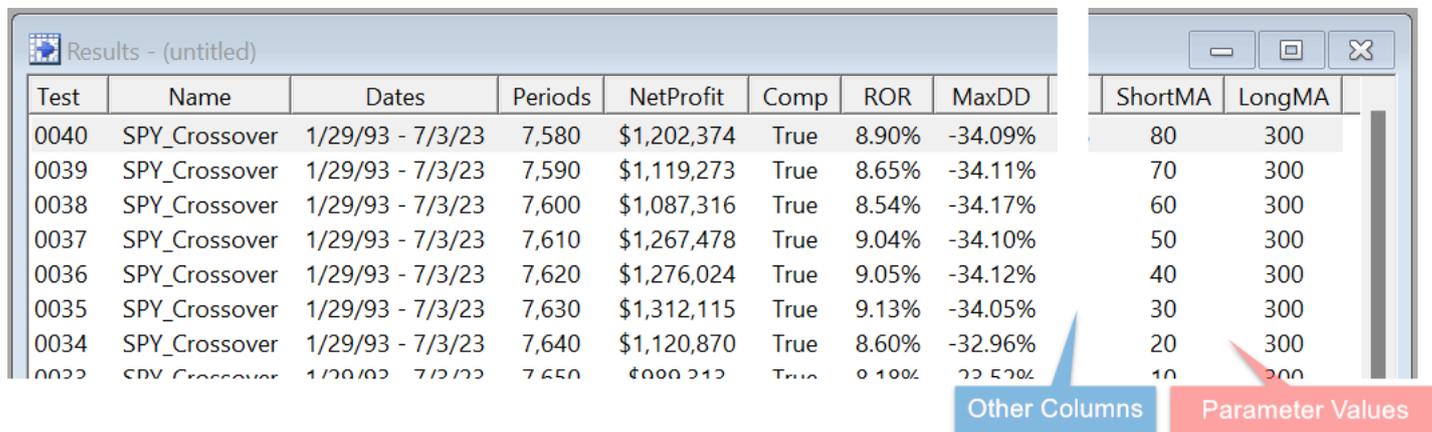


For now, you can ignore most of the settings in this dialog. Just focus on the upper-left corner, where the **Parameters** that were defined in the script are shown. Also notice the lower-right corner, where the number of tests to be run is calculated and displayed.

Click on the check box for each of the two parameters and observe what happens to the Maximum Test Count value. Once both have been checked it should indicate that 40 tests will be performed.

Click  and watch as a new Results Window appears and is quickly populated with test results. (On my machine this takes about 3 seconds.)

Besides the usual columns of the Results Window, notice that two new columns have been added, showing the value of the two parameters for each test.

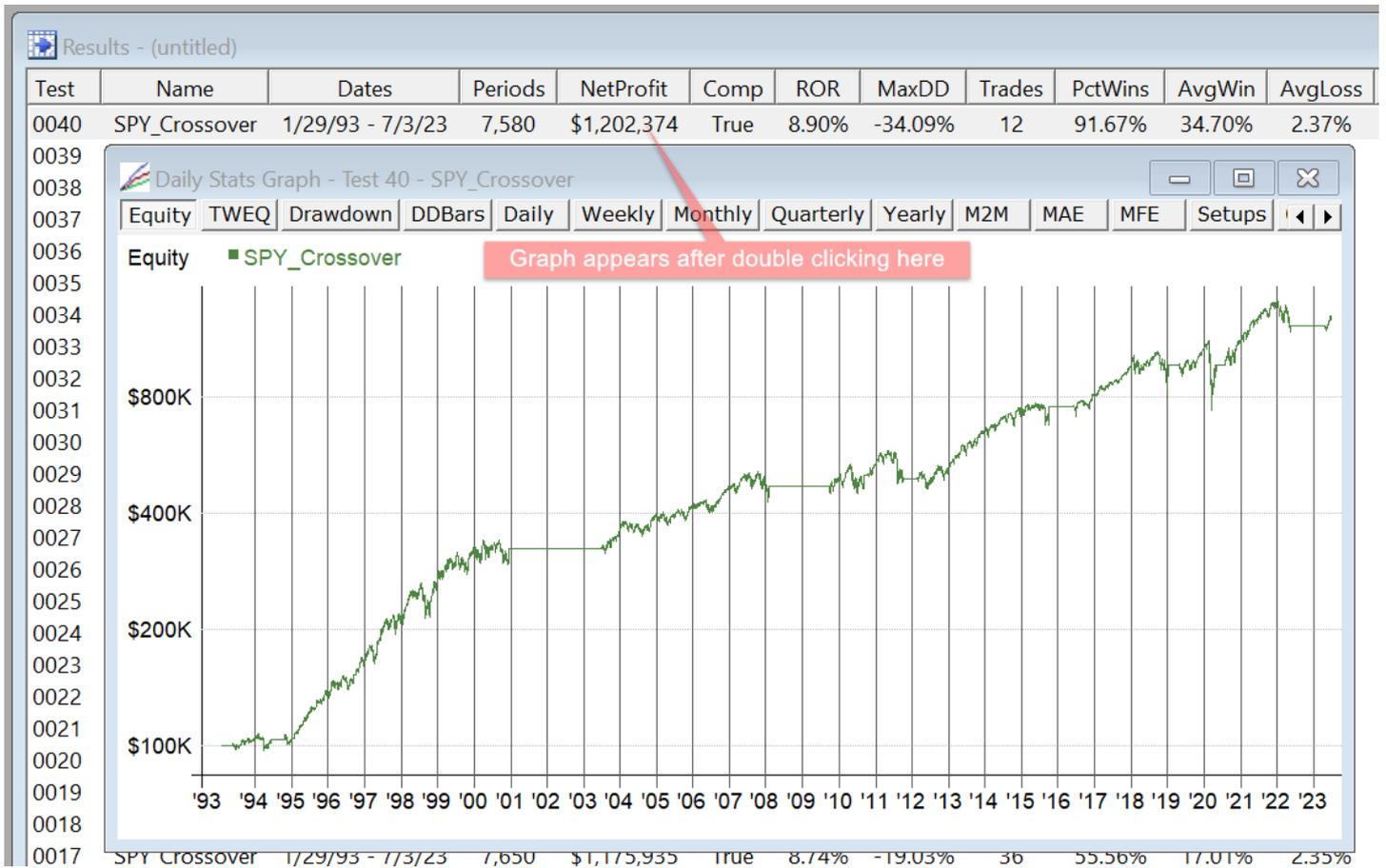


Try clicking on the buttons at the top of various columns in this window and notice that these cause the results to be sorted by that column.

Sorting, for example, by NetProfit can give you a quick idea of which parameters would have done the best.

Click the same column again to reverse the sort order. Shift-click other columns to create a multi-level sort.

After sorting the results list by ascending NetProfit, double-click on the top row to open the Daily Stats Graph showing the equity curve.

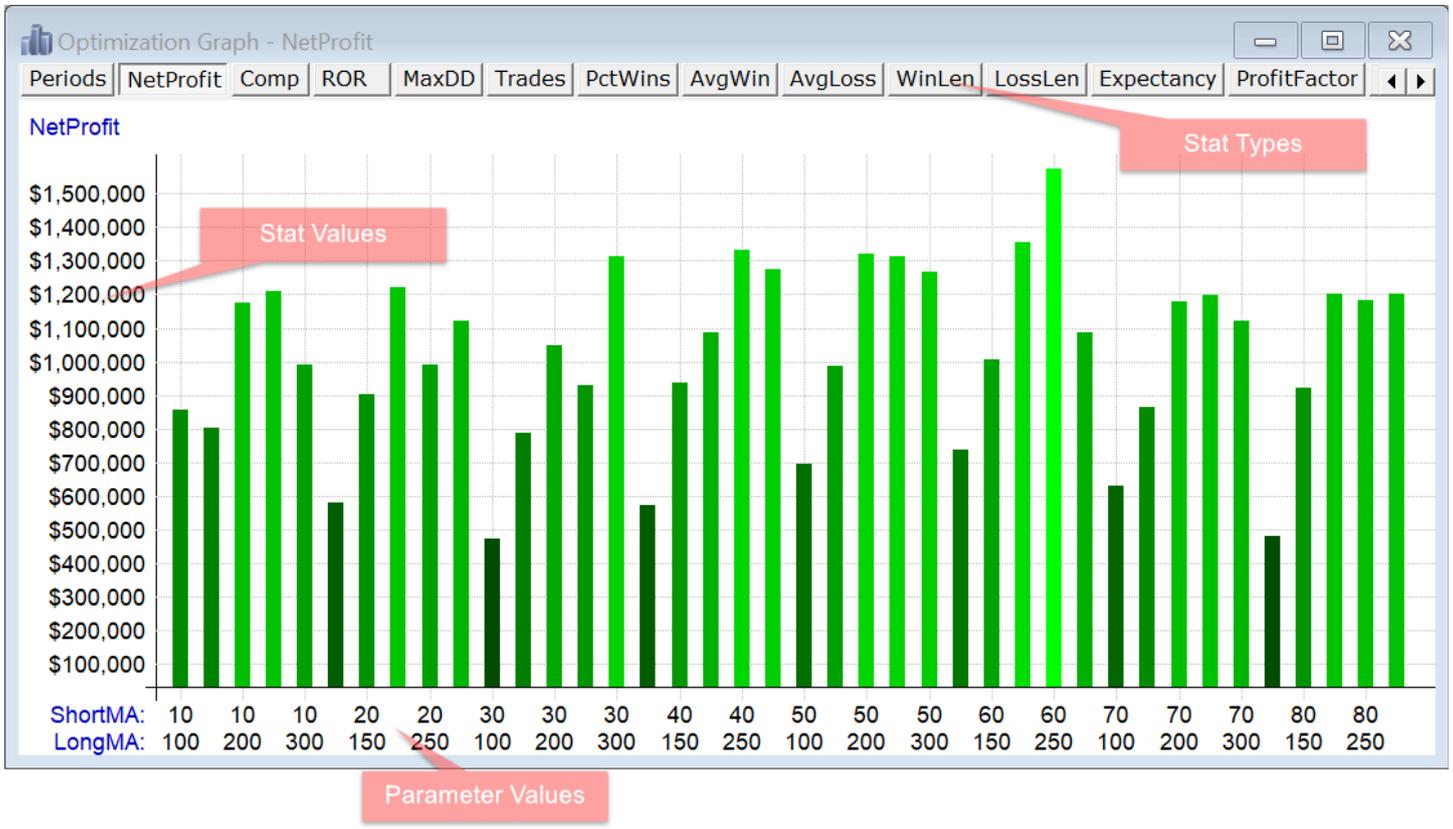


Now repeatedly press the **down arrow key** on your keyboard and watch as the graph changes to show each equity curve from the set of tests. If, as above, you sorted by NetProfit with lowest values first, the curve will gradually look better as you proceed with the down arrow. If you started with highest first, it will gradually look worse. You can even hold down the key and let it auto-repeat to see them all in rapid succession.

Feel free to experiment with the buttons along the top of the graph (or the left and right arrow keys).

To get a better sense of the relationship between these parameters and the corresponding test results,

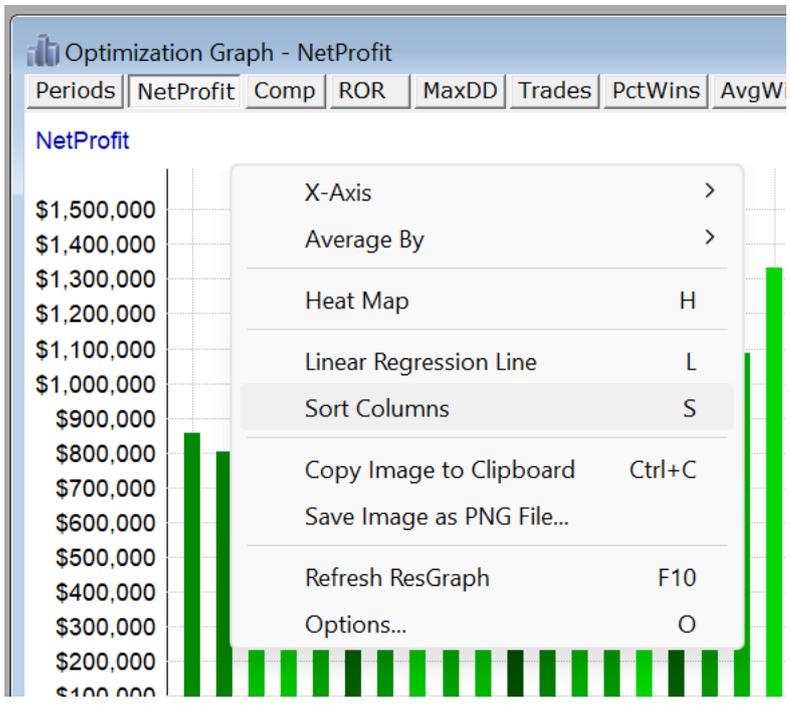
click on  Results in the Tool Bar to open the **Optimization Results Graph**.



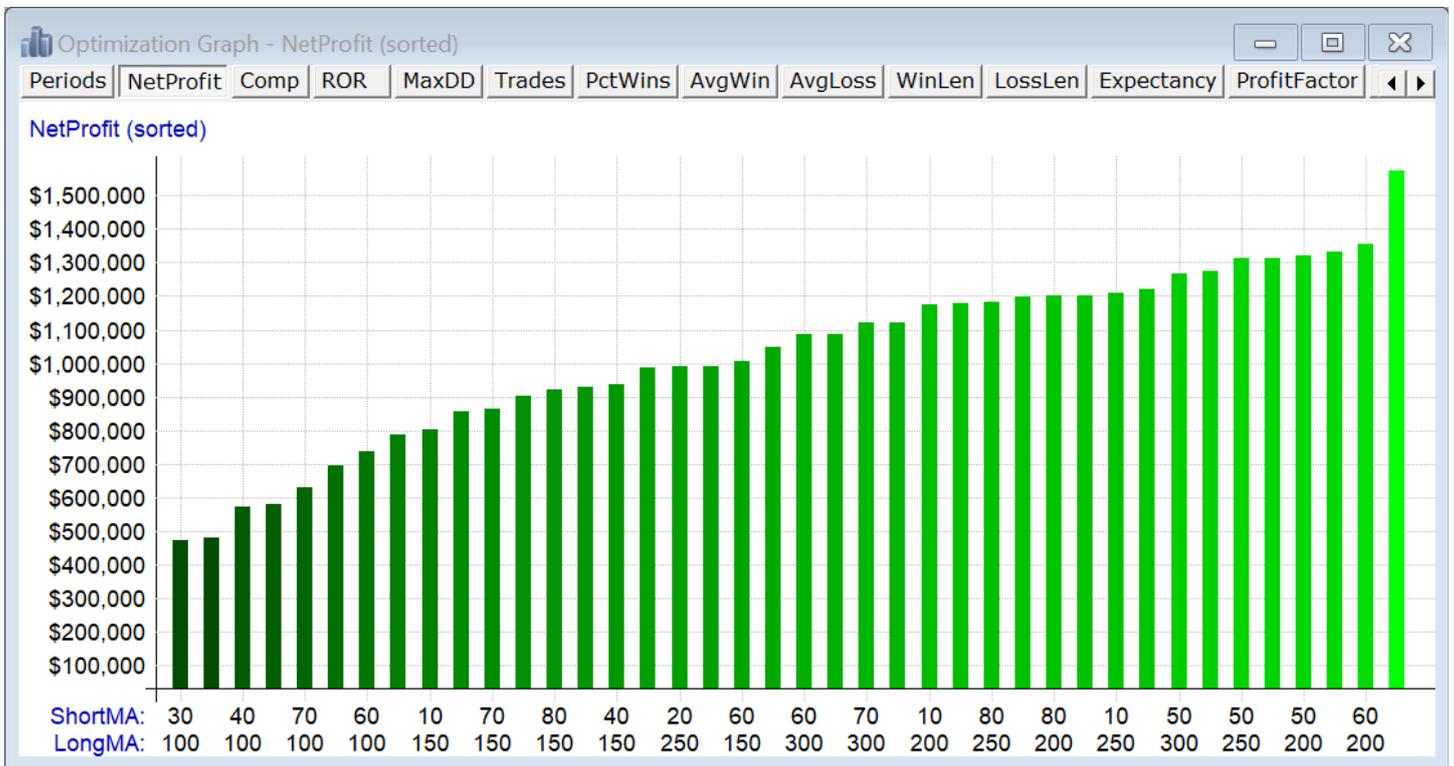
If you see a graph other than NetProfit, use the button bar to select NetProfit. Every column from the Results window can be graphed here.

Note that the X-Axis shows values for both parameters under each bar (or, in this example, under every other bar -- the window would need to be made wider to see every bar label).

Pressing the right mouse button within the graph opens a menu that can be used to change the display in many ways.



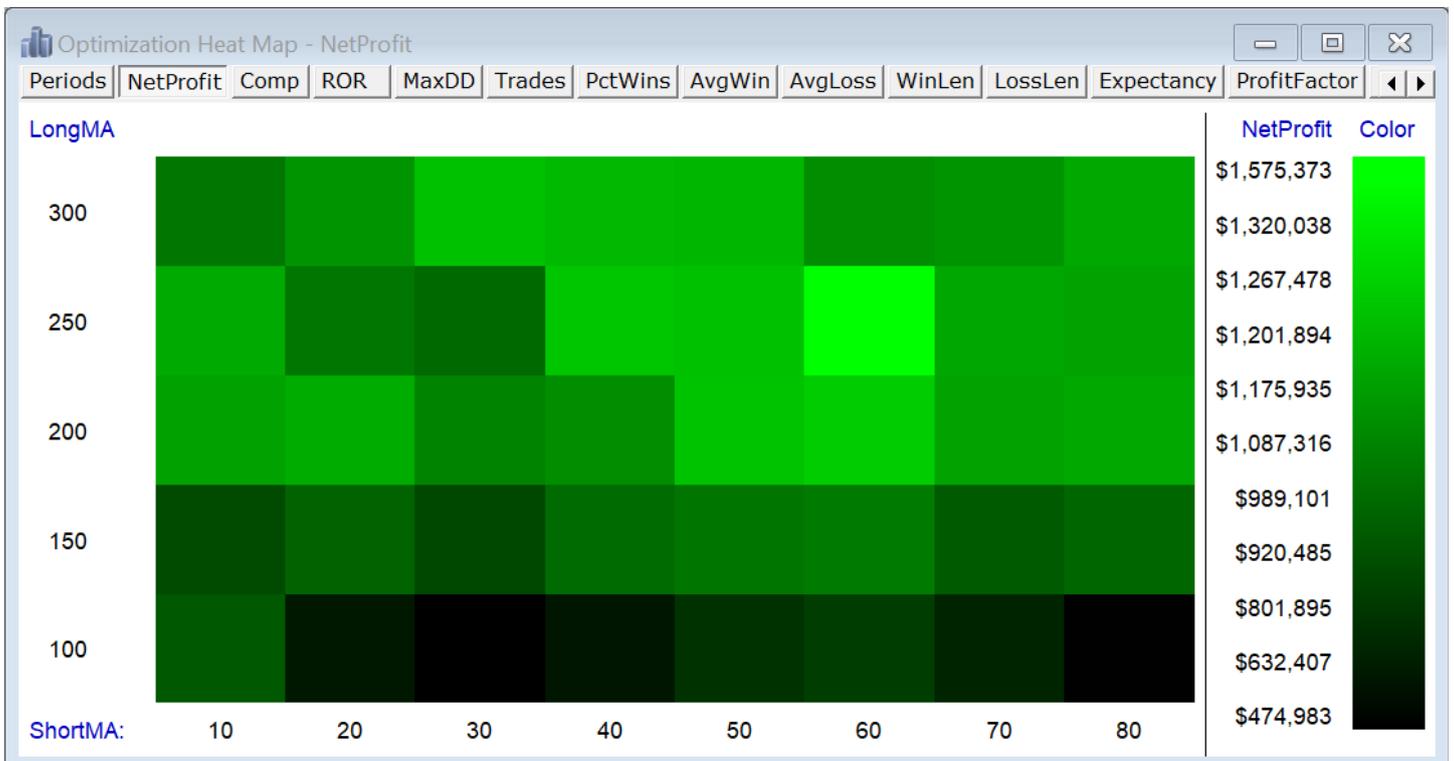
Selecting *Sort Columns* changes the graph to this:



Now we can easily see that 60/200 had the highest net profit of the combinations we tested and that it appears to have been an outlier.

Go back into the popup menu and select **Heat Map**.

Now you'll see this nice-looking checkerboard:



Just another view of the same data, of course, but this makes it easier to see where the best results tend to cluster (in this case, the larger values for Long MA).

(Some people like to see two-parameter optimization results displayed in a rotating 3D graph. Though these look flashy, they add no new information to what is already discernible from a simple heat map.)

To demonstrate the remaining capabilities of the optimization graph, please close all open windows and open the example script called **Sample2a.rts**.

Active Script - C:\RealTest\RELEASE\Examples\Sample2a.rts

Notes: 2-Parameter optimization example  
find out which crossover MA lengths would have done best

Settings: DataFile: sample1.rtd // see Sample1.rts for import instructions  
StartDate: Earliest  
EndDate: Latest

Parameters: ShortMA: 5, 10, 15  
MidMA: 30, 50, 70  
LongMA: 100, 200, 300

Strategy: **SPY\_Crossover**  
EntrySetup:  $\text{Avg}(C, \text{ShortMA}) > \text{Avg}(C, \text{MidMA})$  and  $\text{Avg}(C, \text{MidMA}) > \text{Avg}(C, \text{LongMA})$   
ExitRule:  $\text{Avg}(C, \text{ShortMA}) < \text{Avg}(C, \text{MidMA})$  or  $\text{Avg}(C, \text{MidMA}) < \text{Avg}(C, \text{LongMA})$

Run this script in **Optimize** mode, with all three parameters checked, and then open the **Optimization Graph**.

It will open as a **Heat Map**, since that's the previous view it was shown in.

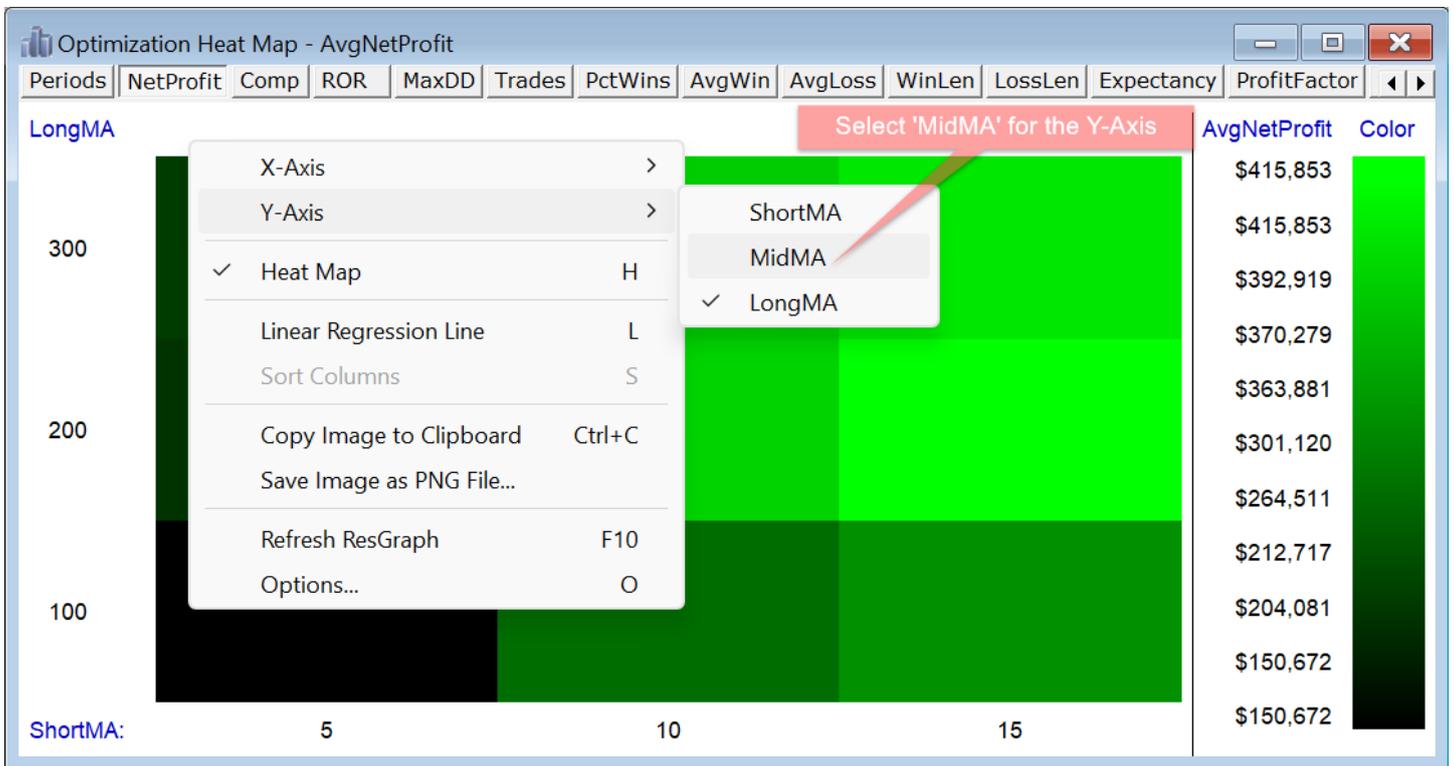


There is one key difference as highlighted above.

We can only show two parameters (as would also be true in a 3D graph), so now the *Average* result (Net Profit in this case) is graphed for all tests with each shown parameter pair.

In this example, each NetProfit value is the average of the three different MidMA values for each pair of LongMA and ShortMA.

Use the right-click popup menu to change the selection of which two parameters to graph.



Now the heat map looks like this:



Now the NetProfit values are the average of the three different LongMA values for each pair of MidMA and ShortMA.

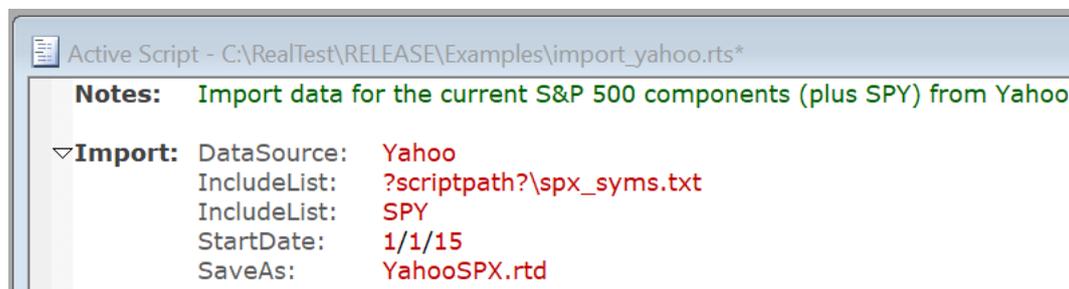
That's the end of this second tutorial!

For information about all of the other features of the RealTest Optimizer, see the [Optimization Dialog](#) topic.

## 6.3. Tutorial 3 - Simple Scan

To start the third tutorial, close any child windows that remain open, then open the script called **import\_yahoo.rts** in the Examples folder (see the [previous tutorial](#) for instructions on how to do

that).

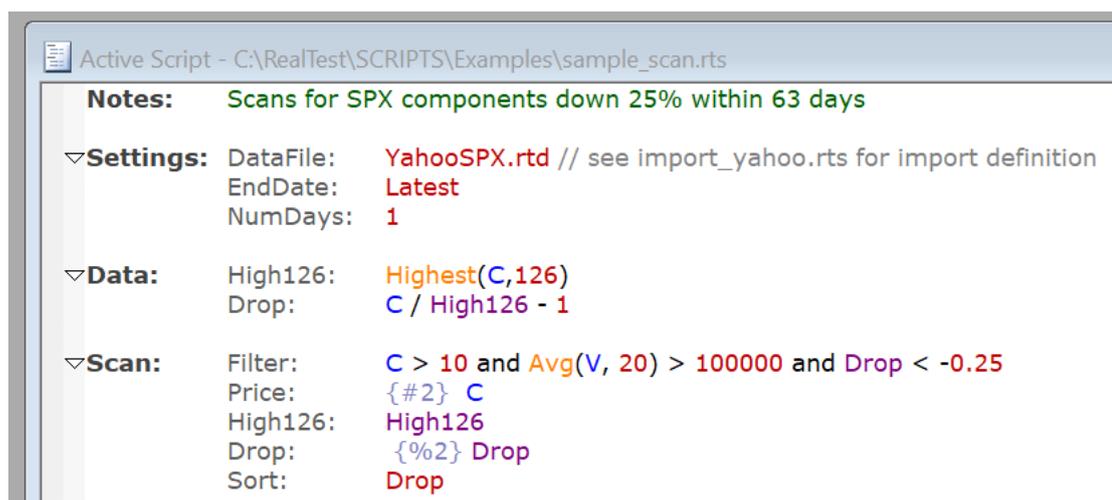


This example script contains only an **Import section**.

Click on  **Import** to run the script and import data for each current (as of June 2022) stock in the S&P 500 index, along with the SPY ETF data, from 2015 through today. (Feel free to edit the SP500.txt file if the components have changed, or even better, use Norgate Data instead of Yahoo to automatically gain access to both current and historical index members.)

You will see that the Yahoo import runs fairly quickly. This is because RealTest creates multiple threads for the import, each with a separate HTTP connection to the Yahoo servers.

When the import has finished, close this script window and open the example script called **sample\_scan.rts**.



This script introduces two new sections: **Data** and **Scan**.

The Data section is probably the most important feature of RealTest. Please take a moment to read about the **Data Section** before continuing.

Did that make sense? Good. Now please take a moment to read about the **Scan Section** as well.

Now we are ready to simply run this scan.

Press the  **Scan** button and the scan output will appear:

Date	Symbol	Price	High126	Drop ▲
7/3/23	AAP	70.95	156.84	-54.76%
7/3/23	ZION	28.14	54.79	-48.64%
7/3/23	CMA	44.00	76.94	-42.81%
7/3/23	ETSY	85.40	148.20	-42.38%
7/3/23	CTLT	44.29	74.26	-40.36%
7/3/23	CFG	26.52	44.46	-40.35%
7/3/23	EPAM	226.58	379.34	-40.27%
7/3/23	VFC	19.14	31.64	-39.51%
7/3/23	MRNA	121.73	197.02	-38.21%
7/3/23	TFC	31.39	50.39	-37.71%
7/3/23	MOS	35.74	57.14	-37.45%
7/3/23	ENPH	169.55	264.96	-36.01%
7/3/23	OGN	20.57	32.08	-35.88%
7/3/23	PARA	16.32	25.25	-35.37%
7/3/23	MKTX	262.15	392.29	-33.17%
7/3/23	SCHW	57.72	85.62	-32.59%
7/3/23	USB	33.58	49.80	-32.57%
7/3/23	PENN	24.27	35.85	-32.30%
7/3/23	ARE	116.12	170.82	-32.02%
7/3/23	IFF	79.73	116.70	-31.68%
7/3/23	DG	170.57	248.56	-31.38%

In this example, we are scanning only the most recent date in the data for stocks that are currently down at least 25% from their 126-day (half-year) highest close.

At the time of this writing, 38 stocks met this condition.

Now double-click on the first row to open a chart for that stock with that date as its right-most bar:



Once the chart window is open, you can use the **down arrow key** on your keyboard to cycle through

all the charts from the scan output (or press up to go back to the previous one). Notice that the scan window automatically highlights the line corresponding to the current chart symbol. To see both at once, just move and resize the two windows as desired.

To see other things that can be done with scan output, open the **Scan Menu** either from the menu bar (when the scan is the active window) or by right-clicking within the scan window:

Date	Symbol	Price	High126	Drop ▲
7/3/23	AAP	70.95	156.84	-54.76%
7/3/23	ZION	28.14	54.79	-48.64%
7/3/23	CMA	44.00	76.94	-42.81%
7/3/23	ETS			
7/3/23	CTL			
7/3/23	CFC			
7/3/23	EPA			
7/3/23	VFC			
7/3/23	MRM			
7/3/23	TFC			
7/3/23	MOS	35.14	57.14	-37.45%

Context menu options for CMA:

- Show Chart (Enter)
- Debug Item (Ctrl+Enter)
- Save All Charts...
- Copy List to Clipboard (Ctrl+C)
- Save List as CSV File...

The entire contents of the scan can be copied to the clipboard (from which it will paste into Excel with columns preserved) or saved as a CSV file.

There is also a **SaveScanAs** option in the **Settings** section, which, if specified, causes the scan output to be automatically saved every time the scan is run. This can be useful for something like producing a daily trading candidate list.

That's the end of this third tutorial!

## 6.4. Tutorial 4 - ETF Rotation

In this next tutorial, a more elaborate backtest script will be introduced.

Please close all currently open windows, and then open the file **sector\_etfs.rts** from the Examples directory.

```

Active Script - C:\RealTest\SCRIPTS\Examples\sector_etfs.rts

Notes:      S&P sector ETF monthly rotation based on n-day ROC

▼Import:    // Import the S&P sector ETFs since 1999 from Yahoo
DataSource: Yahoo
IncludeList: XLB, XLE, XLF, XLI, XLK, XLP, XLU, XLV, XLY {"sectors"}
IncludeList: SPY // for benchmark
StartDate:  1/1/1999
SaveAs:     sector_etfs.rtd

▼Settings:  DataFile:    sector_etfs.rtd
            StartDate:  Earliest
            EndDate:    Latest

▼Parameters: // first value is the default for single test mode
Lookback:   63, 42, 21, 126, 252
Positions:  5, 4, 3, 2, 1
Momo:       1, 0
  
```

The above is, of course, a partial screenshot. This script is longer than the prior examples, and there's a lot going on, so take some time to study it carefully.

This example contains elements introduced in prior tutorials, including **Import**, **Settings**, **Parameters**, **Data** and **Strategy** sections.

Two new sections are introduced here, which are both derivations of Strategy: **Template** and

## Benchmark.

The *Strategy* section is where a trading strategy is defined for the purpose of backtesting it, as seen in the first two tutorials.

The *Template* section is an optional element that can be used to avoid repeating the same elements in multiple strategies. In other words, it is where you might define any elements that are common to more than one strategy. In this example, the Template called "base" is only used by one strategy, so it doesn't serve any purpose besides illustrating what a Template is.

In fact, this part of the script:

```
▼Template: base // common strategy elements
    Side: Long
    Quantity: 100 / Positions
    QtyType: Percent
    MaxInvested: S.Equity
    MaxPositions: Positions
    Commission: 0.005 * Shares
    Slippage: 0.001 * FillPrice

▼Strategy: sector_etfs
    Using: base // inserts those extra elements into this strategy
    EntrySetup: Rotate and MyRank <= Positions
    ExitRule: Rotate and MyRank > Positions
```

could just as easily have been written as follows, with the template omitted:

```
▼Strategy: sector_etfs
    Side: Long
    Quantity: 100 / Positions
    QtyType: Percent
    MaxInvested: S.Equity
    MaxPositions: Positions
    Commission: 0.005 * Shares
    Slippage: 0.001 * FillPrice
    Using: base // inserts those extra elements into this strategy
    EntrySetup: Rotate and MyRank <= Positions
    ExitRule: Rotate and MyRank > Positions
```

The two snippets above are functionally identical. The **Using: base** statement in the first snippet instructs the script parser to copy all the elements from the base template and insert them in the `sector_etfs` strategy. The idea behind this is to use "Template: base" for strategy elements such as Commission and Slippage that are often the same for every strategy.

In contrast to *Template*, a *Benchmark* section defines a strategy that will be included as a separate entity in the backtest. It is "traded" for each date in the test just as a regular strategy is and will generate its own statistics and be given its own equity curve. The only difference between a benchmark and a strategy is that the benchmark's stats are not counted as part of the combined results of the backtest.

In this script example, the benchmark is used to simply generate an equity curve for the SPY ETF in a buy-and-hold simulation, to make it easy to visually compare the strategy's equity curve to this common benchmark:

```
▼Benchmark: benchmark
    Side: Long
    EntrySetup: Symbol=$SPY
    // exit (and immediately re-enter) on each ex-dividend day, to re-invest the dividend
    ExitRule: Dividend > 0
```

(Because Yahoo data is not dividend-adjusted but does include specific dividend amounts and dates, in order for the SPY dividends to be re-invested, the benchmark simulates exiting and re-entering the SPY position on each morning after an ex-dividend date.)

Now on to the specific strategy and its implementation in this script.

The concept is to trade the 9 standard sector ETFs derived from components of the S&P 500 index (XLB, XLE, XLF, XLI, XLK, XLP, XLU, XLV, XLY) using a monthly rotational rule based on momentum. The Data section defines all the variables needed to implement this strategy. Parameters are used in

data formulas to enable some optimization.

```

▽Parameters: // first value is the default for single test mode
Lookback:    63, 42, 21, 126, 252
Positions:   5, 4, 3, 2, 1
Momo:       1, 0

▽Data:      Strength:    C / Lowest(C, Lookback)
MomoRank:   #Rank if(InList("sectors"), Strength, nan)
ValueRank:  #Rank if(InList("sectors"), 1 / Strength, nan)
MyRank:     IF(Momo, MomoRank, ValueRank)
Rotate:     EndOfMonth // tomorrow is a new month

```

- *Strength* is simply the ratio of the current close to the lowest close in some number of days (63 by default, i.e., one calendar quarter)
- *MomoRank* uses the cross-sectional rank function to calculate the strength's rank for each ETF for each date relative to the other ETFs on that date
- *ValueRank* is the inverse of *MomoRank*
- *MyRank* selects which of the above two ranks to use depending on the *Momo* parameter (allows comparison of momentum vs. mean-reversion as the ranking criterion)
- Note also that these two rank calculations deliberately exclude the SPY benchmark from the top ranks by checking the **InList** name in the rank formula
- *Rotate* evaluates to TRUE (1) for the last trading day of each month

The strategy is defined to use the above data items as follows:

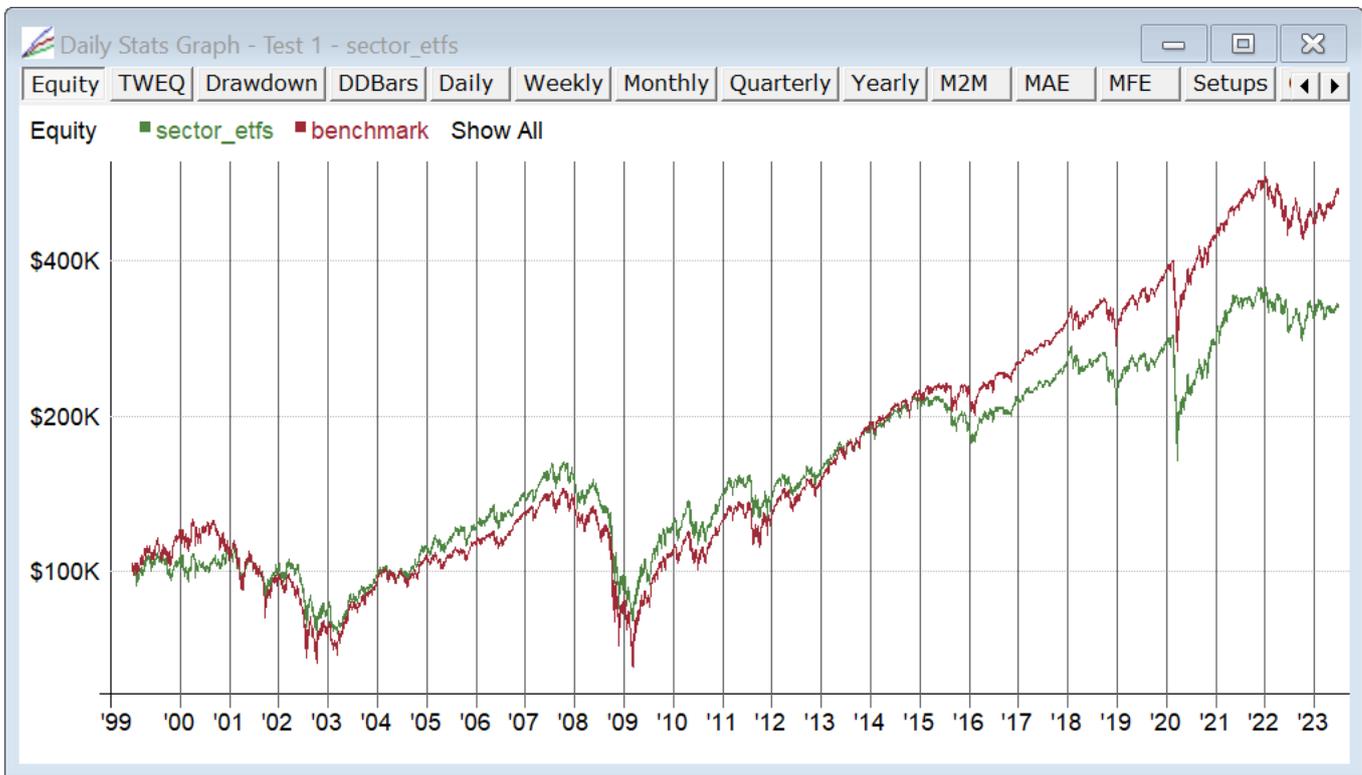
- **EntrySetup** must evaluate to TRUE (1) for there to be an entry at the next open. It will evaluate to TRUE for each ETF that is not SPY and whose rank is low enough (1 is top-ranked) to be within the **MaxPositions** threshold. All this will only occur if the next open will be the first open of a new month. The three clauses of the *EntrySetup* formula and the data elements that they refer to suffice to specify all of these factors.
- **ExitRule** is one of several exit elements a strategy can have (in this case it's the only one). This rule simply says to exit if it's a new month and the ETF is no longer among the top ranked ETFs.
- **Quantity** specifies the position size, in percent of current equity. **S.Equity** is the current account balance (including mark-to-market, i.e., net liquidation value) on any given date during the test, so this formula is simply a percentage of equity based on the *Positions* parameter.

To see all of this in action, first run this script first as  and then as  as we did in the first tutorial.

A results window will appear with a new line for the test results (yours will be slightly different):

Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	Trades	PctWins	AvgWin	AvgLoss
0001	sector_etfs	1/4/99 - 7/3/23	6,145	\$232,099	True	5.04%	-50.98%	405	51.11%	9.18%	5.77%

Not very promising. Let's open the equity graph to see how it looks in more detail (double-click on the above row):



(Note that you can use the L key to toggle between Log scale and Arithmetic scale, or press the right mouse button on the graph to see this and other options on its menu.)

This graph shows the purpose of the SPY benchmark. Buy and hold SPY is not included in the reported 5.04% ROR or 50.98% MaxDD. Those are the stats for the ETF strategy alone. But the SPY benchmark is included in the above graph as the red line, showing that, overall, this ETF rotational strategy did not beat buy-and-hold of the index, though prior to 2014 it was slightly ahead.

Since we have some parameters, we can optimize them to see how that will affect the results. Click on **Optimize** and then just select the Lookback parameter for now:

RealTest Optimizer

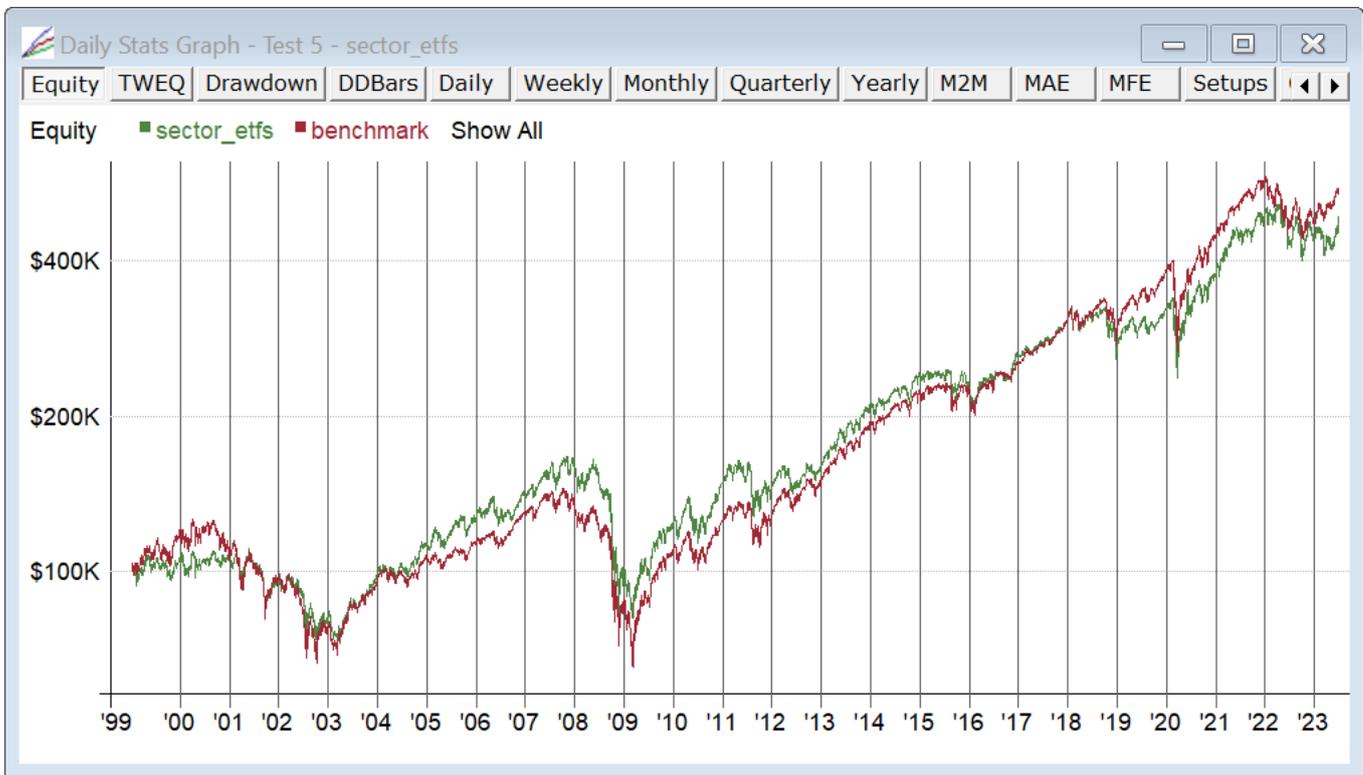
Parameter	Count	Values (first is default)
<input checked="" type="checkbox"/> Lookback	5	63, 42, ..., 126, 252
<input type="checkbox"/> Positions	5	5, 4, ..., 2, 1
<input type="checkbox"/> Momo	2	1, 0

Run the 5 tests and look at the results:

Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	Lookback	Positions	Momo
0005	sector_etfs	1/4/99 - 7/3/23	6,145	\$390,716	True	6.73%	-51.44%	252	5	1
0004	sector_etfs	1/4/99 - 7/3/23	6,145	\$333,465	True	6.19%	-53.48%	126	5	1
0003	sector_etfs	1/4/99 - 7/3/23	6,145	\$220,474	True	4.89%	-50.20%	21	5	1
0002	sector_etfs	1/4/99 - 7/3/23	6,145	\$156,440	True	3.93%	-50.14%	42	5	1
0001	sector_etfs	1/4/99 - 7/3/23	6,145	\$232,099	True	5.04%	-50.98%	63	5	1

Other Columns Here

Double-click on the top row (252 bars, a one-year lookback) to show how this parameter has slightly out-performed one we tested initially (63 bars, a one-quarter lookback):



Now if you want to go nuts with data mining, go back into Optimize and check all 3 parameter boxes. Run all 50 tests and sort by net profit to see what would have worked best:



Now the rotation is substantially beating the benchmark. But which parameters were the winners?

Lookback	Positions	Momo
42	4	0

Apparently relative weakness (mean reversion) did better than relative strength (momentum) in this

case.

In fact, the top 17 results are all using this anti-momentum factor:

Lookback	Positions	Momo
42	4	0
42	3	0
42	5	0
63	3	0
63	4	0
42	2	0
126	3	0
63	2	0
126	4	0
21	5	0
21	3	0
63	5	0
252	5	0
21	4	0
126	2	0
252	4	0
126	5	0

This is a good example of how we can sometimes learn something from an optimization, even if the goal is not (and should never be) to over-fit the parameters to the data for actual trading.

This is the end of the fourth and final written tutorial in this User Guide.

A series of video tutorials can be found on the [mhptrading youtube channel](#).

# 7. Software User Interface

RealTest uses the classic Windows **Multiple Document Interface** paradigm.

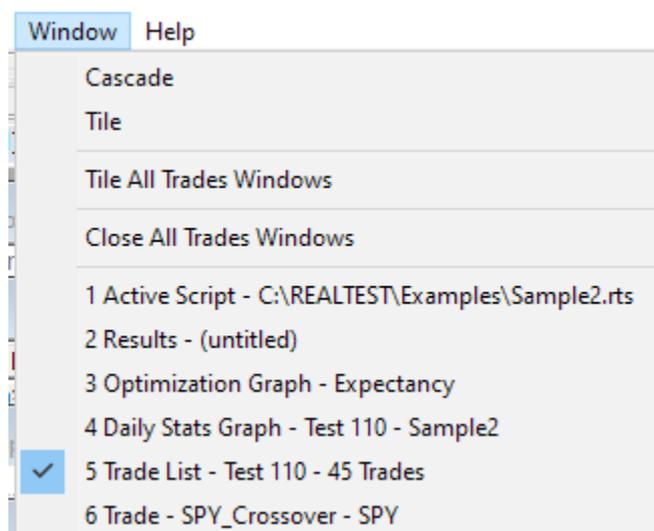
The main window contains a variety of child window types, and the main window's menu bar changes depending on which type of child window is currently selected.

The child-specific menu is also always available as a popup menu by pressing the right mouse button within the child window.

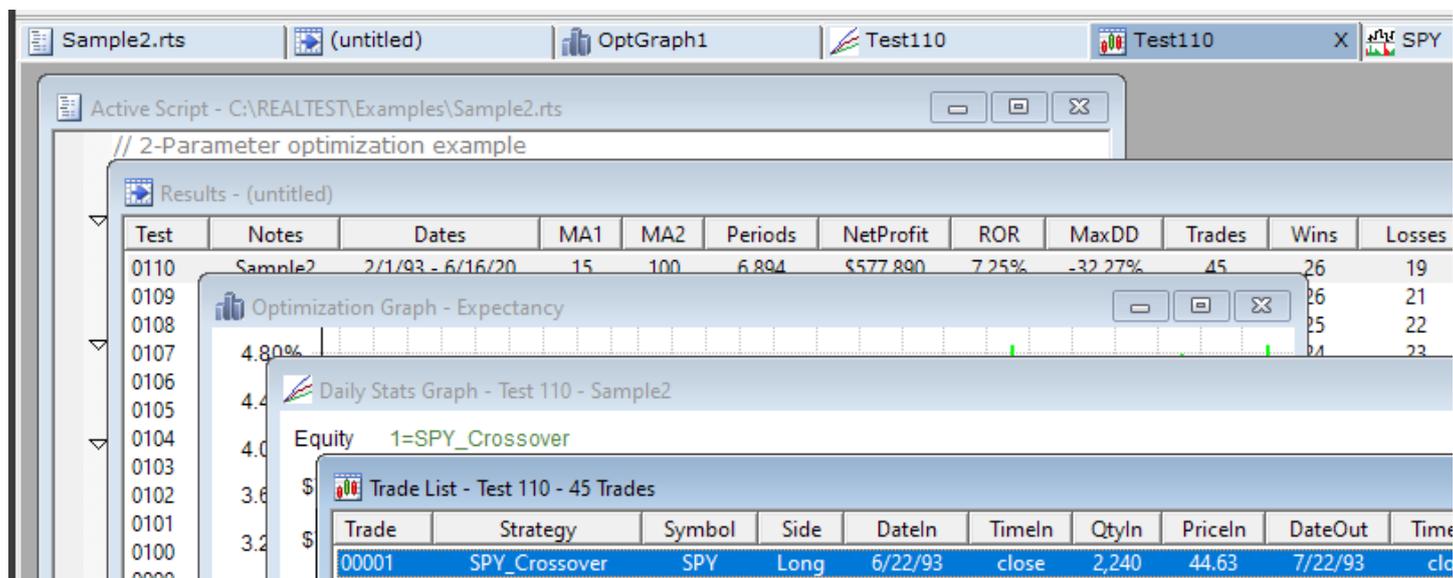
Child windows within the main window can be arranged manually or by using the commands on the Window menu.

Child windows can be maximized or minimized as needed.

Any specific window can be selected and activated using the Window menu as well.



To save you from always having to open the Window menu to activate a different child window, RealTest also provides **Window Tabs**. Click on a tab to activate that window, double-click to maximize or restore the window, or use the [X] to close the window. Drag the tabs to change their order if desired.



Child window/document types include:

- **Scripts**
- **Results** (columnar list of tests run with summary stats)
- **Stats Graphs** (daily stats from a test)

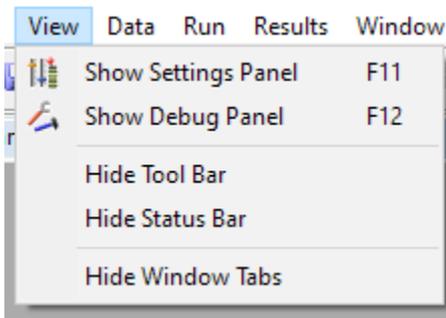
- **Optimization Graphs** (bar chart or heat map of optimization results by parameter)
- **Trades** (list of trades from a test)
- **Trade Plots** (scatter plot, trade-level profit curve, Monte Carlo analysis, etc.)
- **Scan** (list of date/symbol / custom data output from a scan)
- **Charts** (bar or candlestick chart from a trade list or scan)
- **Lists** (general-purpose tabular data display)
- **Logs** (optional text output from testing, analysis or debugging)

In addition to these child windows, there are two permanent dialog boxes:

1. **Settings Panel** (shown above)
2. **Debug Panel** (not shown above)

Each of these two dialogs can easily be hidden or shown using the "View" menu and/or dedicated function keys (F11 and F12).

The Window Tabs, Tool Bar and Status Bar can also be optionally hidden or shown using the View menu.



## 7.1. Common Menus

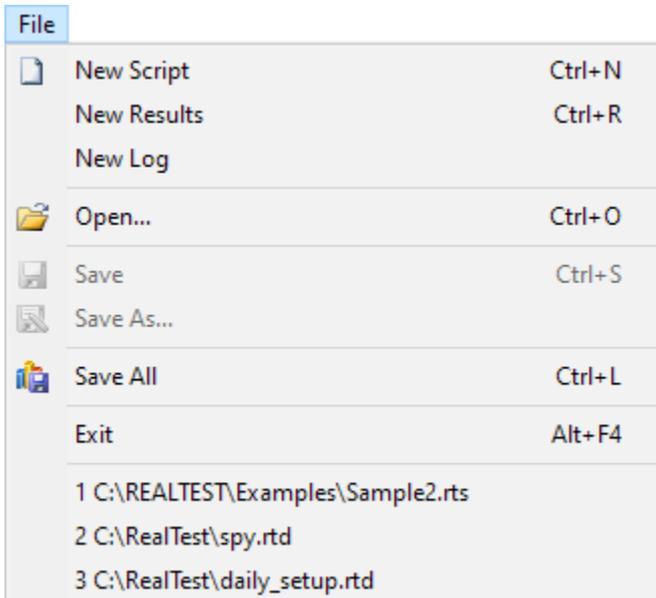
---

The menu bar items described in this section are always present, no matter what type of child window is currently active.

### 7.1.1. File Menu

---

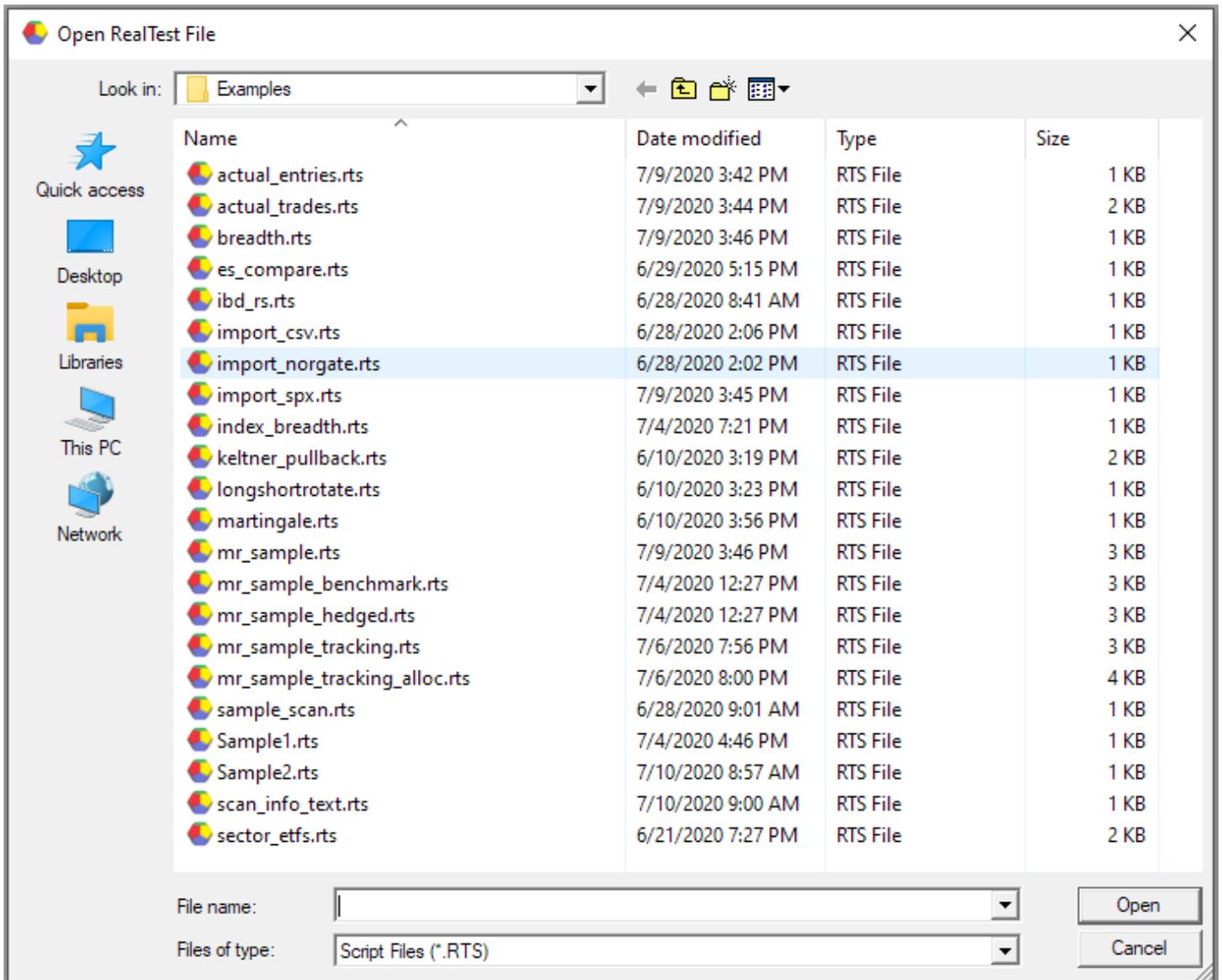
The *File* menu, as in most software, is used to open or save files, or to exit the program.



Menu items with icons next to them are also accessible using the Tool Bar.

A list of the 9 most recently opened files is automatically shown at the bottom of the menu.

Selecting *File / Open...* presents a standard Windows file selection dialog.



The current folder shown in this dialog will always be the one that you last navigated to.

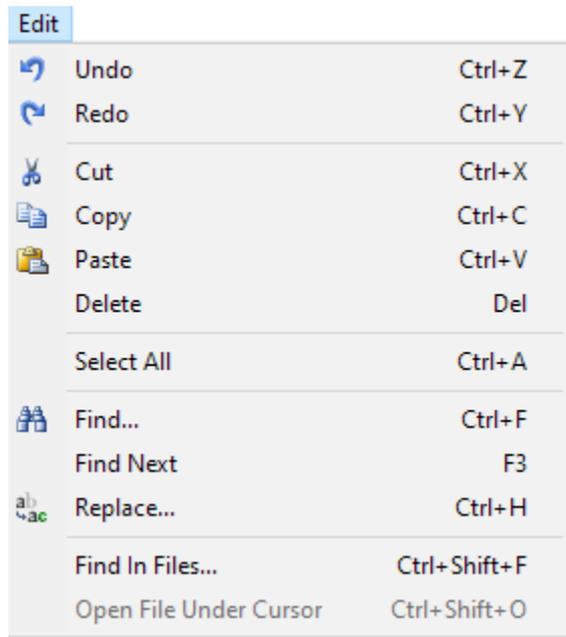
The default file type selection is *Script Files (\*.RTS)*. To see other kinds of files, change this selection.

Drag the lower-right corner of the dialog to change its size. The new size will be remembered next time you select a file.

## 7.1.2. Edit Menu

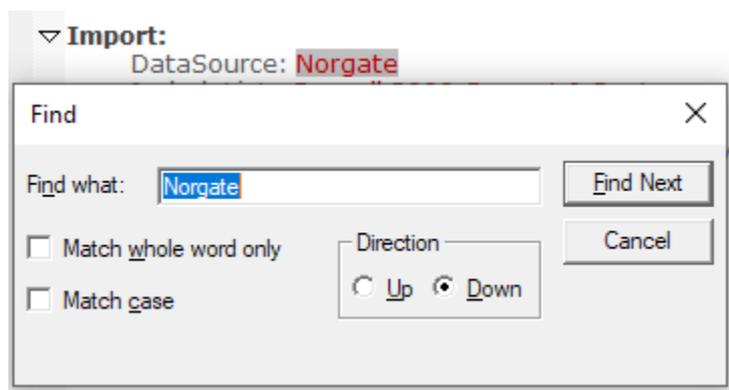
---

The *Edit* menu, as in most software, is used to do something with text in the active window.



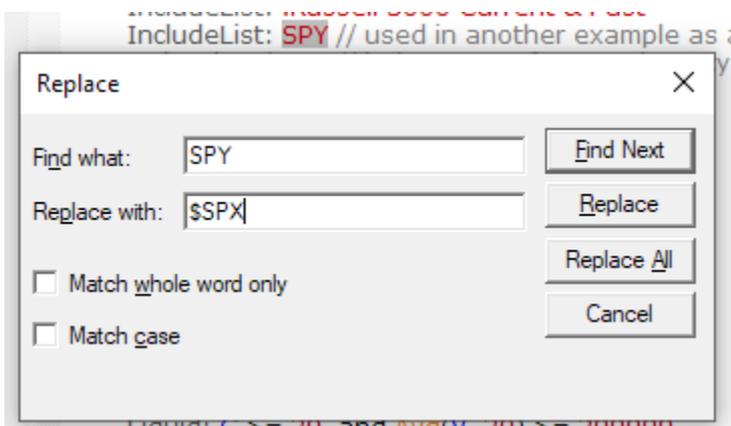
Menu items with icons next to them are also accessible using the Tool Bar.

Use *Edit / Find...* to search for specific text:

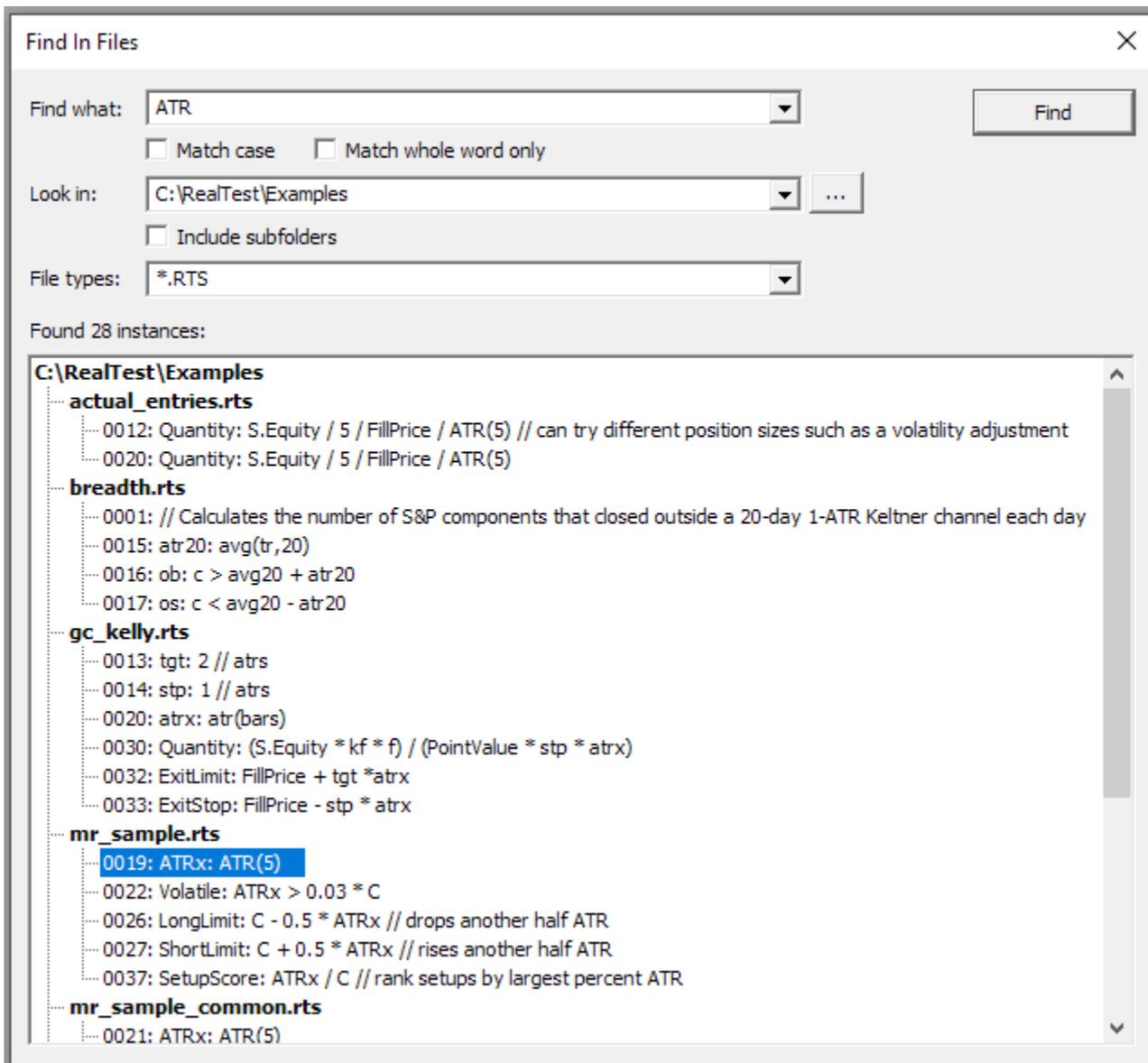


The *Find what* field is automatically filled in with the selected text (or text under the cursor if none is selected) in the editor.

Use *Edit / Replace...* to find specific text and replace it with other text, either once or multiple times:



Use *Edit / Find In Files* to search for text in a collection of files:



Double-click on any line of found text to open that file and highlight that instance of the text:

```

Active Script - C:\RealTest\Examples\mr_sample.rts
// A long/short Russell 3000 mean-reversion strategy pair
// requires Norgate data to test properly

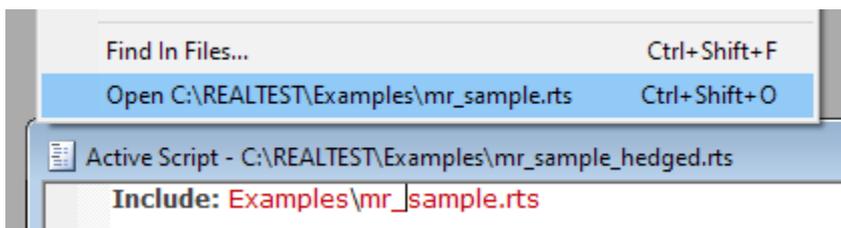
▽ Import:
  DataSource: Norgate
  IncludeList: .Russell 3000 Current & Past
  IncludeList: SPY // used in another example as a hedg
  IndexList: $RUA // index to use for constituency check
  StartDate: 1/2/2010
  EndDate: Latest
  SaveAs: russell3000.rtd

▽ TestSettings:
  DataFile: russell3000.rtd
  StartDate: Earliest
  EndDate: Latest

▽ Data:
  ATRx: ATR(5)
  EMAX: EMA(C,5)
  Liquid: C >= 20 and Avg(V, 20) >= 200000
  Volatile: ATRx > 0.03 * C

```

The last item on the menu, *Open File Under Cursor*, changes to show the path of the specific file that would be opened when the cursor in the active window is within a valid file path:

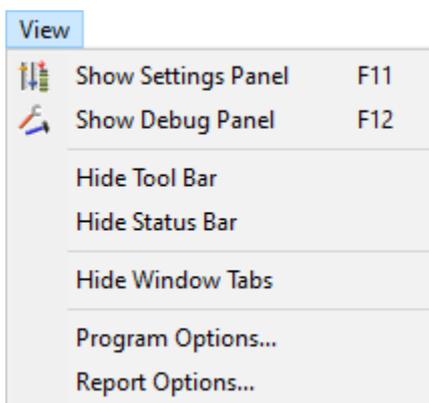


This provides a quicker way to open an included script, a file used in Import, etc. rather than having to select it using *File / Open*.

### 7.1.3. View Menu

---

The *View* menu is used to show or hide specific parts of the RealTest main window.



Menu items with icons next to them are also accessible using the Tool Bar.

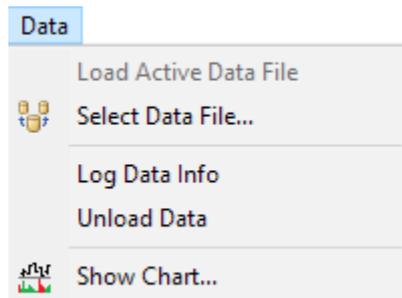
The first five items on the menu function as toggle switches, as reflected by the text of the item saying "Hide" or "Show".

The last two items invoke the **Program Options** and Report Options dialog boxes.

## 7.1.4. Data Menu

---

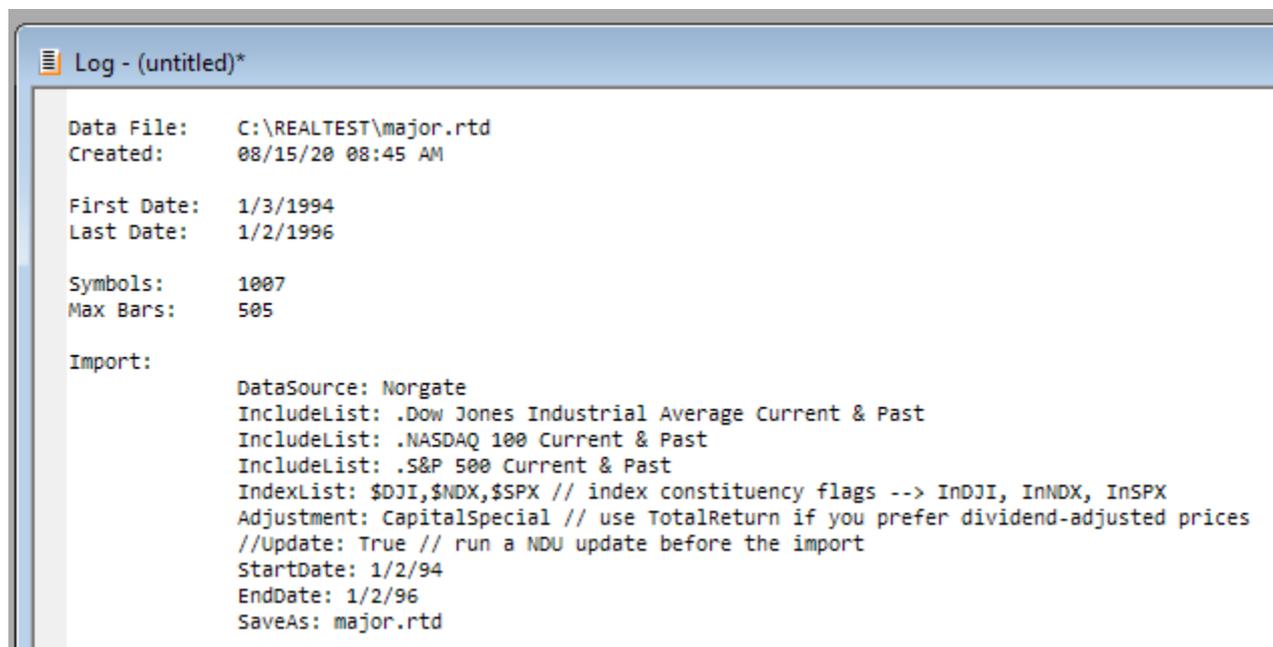
The *Data* menu is used to load or unload the active data file (.RTD) or show a chart.



**Select Data File** opens a file selection dialog to allow selection of the active data file (RTD). Once a data file is selected, the current file (if any) is unloaded from memory and the selected file is loaded into memory. Most of the time, the active data file is loaded automatically when a script is run, so there is rarely a need to use this menu item.

**Log Data Info** writes information about the currently loaded data file to a log window, including the **Import Section** that was used to create it.

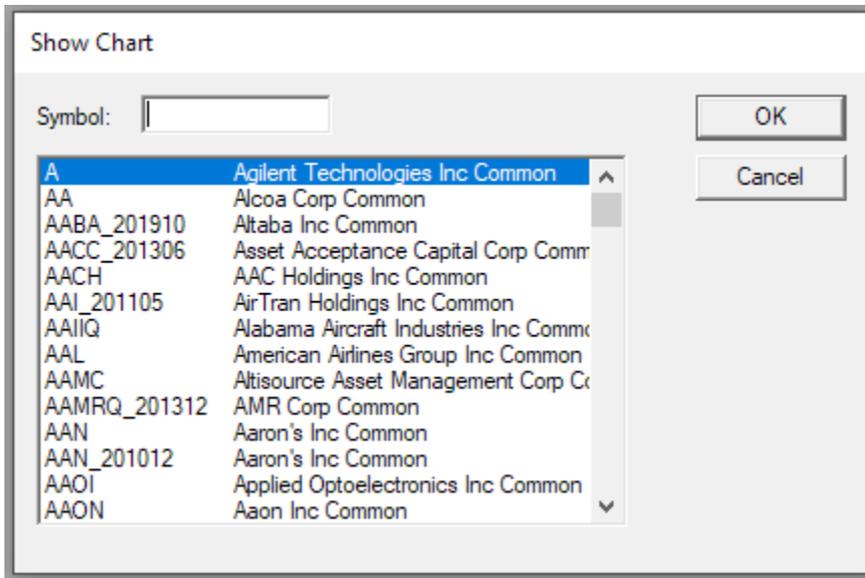
This is an example using the data file created by the **import\_norgate.rts** example script:



**Unload Data** closes all windows that rely on data being in memory (Charts, Scans, etc.) and then releases active data from memory.

**Show Chart** provides a general-purpose way to open a **Chart Window** if a data file has been loaded into memory.

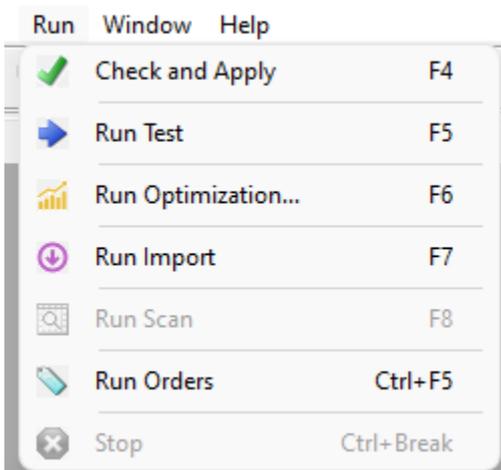
A dialog box is opened with a list of available symbols to choose from:



Once a chart has been opened, the symbol shown can be changed by using the up or down arrow keys or selecting **Jump to Symbol** from the **Chart Menu**.

## 7.1.5. Run Menu

The *Run* menu is used to run the **active script** and select the mode in which to run it.



Menu items with icons next to them are also accessible using the Tool Bar.

The items on this menu are automatically enabled or disabled to reflect the contents of the active script.

**Check and Apply** validates the syntax of the active script. If the script contains one or more sections that define window contents (**Charts, Graphs, Results, Trades**) then it also updates all open windows of those types as specified in the script. Note that it is not necessary to select this item prior to any of the "Run" items -- they all automatically perform the *check and apply* operation before running.

**Run Test** runs all **Strategy** (and Benchmark) sections of the active script as a single test using default parameters. Before the test is run, the **Data** section is recalculated as needed.

**Run Optimization** opens the **Optimization Dialog**, from which one or more tests are run in one of several possible optimization modes.

**Run Import** runs the **Import** section of the active script.

**Run Scan** runs the **Scan** section of the active script.

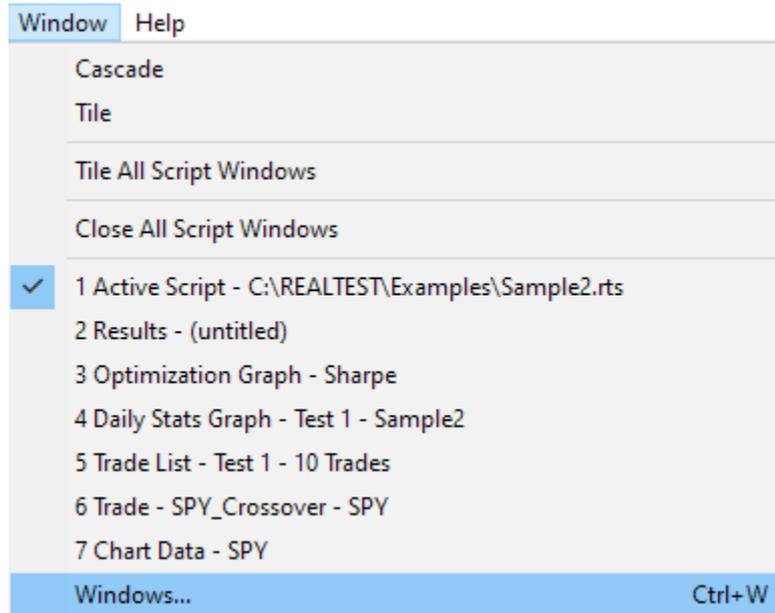
**Run Orders** generates **brokerage orders** for the next market date after the test **EndDate** by first running the test up to that date, then applying the Strategy rules to the data and open positions as of that date.

**Stop** stops whatever operation is currently running.

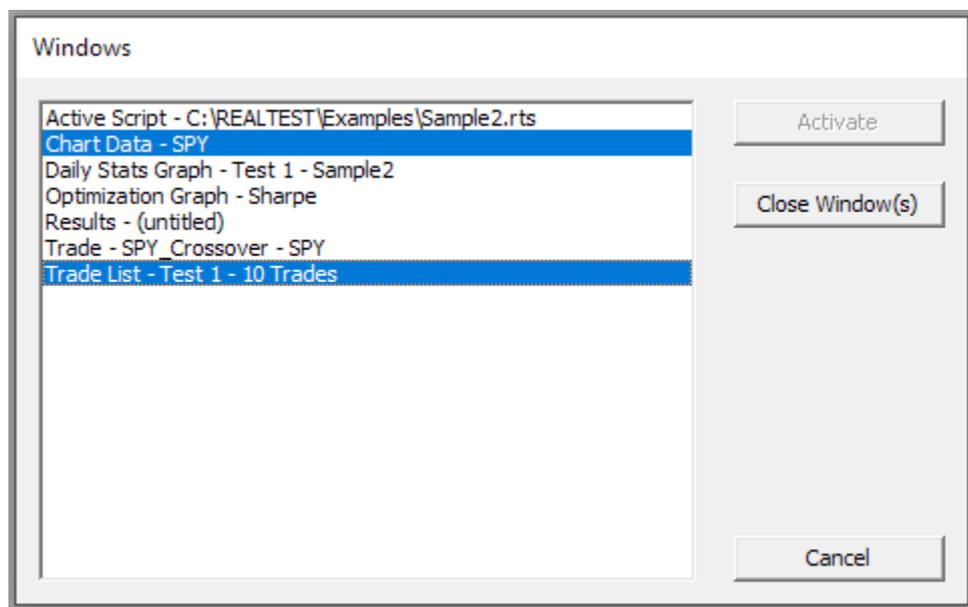
## 7.1.6. Window Menu

---

The *Window* menu is used to change the way child windows are displayed or to activate a specific child window.



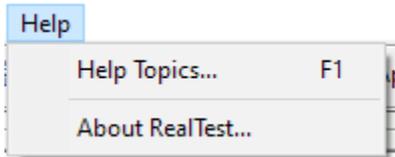
If you open more than 9 windows open and/or want to select several specific windows to close at once, open the *Windows Dialog* from the above menu or by pressing Ctrl+W.



## 7.1.7. Help Menu

---

The *Help* menu is used to open the help file (RealTest User Guide) or the About dialog.



Selecting *Help Topics* (most often done by pressing F1) opens the User Guide and finds the section most relevant to the currently active window.

If a **Script** window is active, pressing F1 will look for a help topic relevant to the currently selected text or the text that the cursor is in.

## 7.2. Child Window Menus

---

The menu bar items described in this section become visible only when the active child window is the corresponding type. For example, the Chart menu appears in the menu bar when a chart is the active window. These menus can also be accessed by pressing the right mouse button anywhere within the child window.

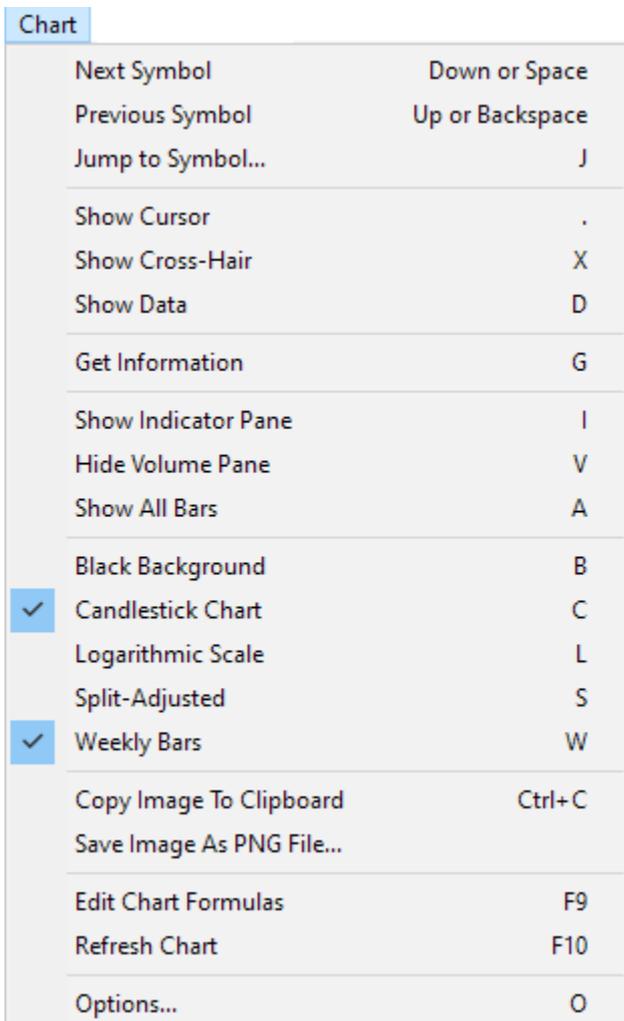
### 7.2.1. Chart Menu

---

The *Chart* menu appears in the menu bar whenever a **Chart Window** is active, and can also be accessed by pressing the right mouse button within that window.

Each item on this menu also has a single keystroke associated with it, as shown to the right of the item.

Once you become familiar with these **keyboard shortcuts**, it is rarely necessary to open this menu.



**Next Symbol** and **Previous Symbol** can be used to quickly move through the list of symbols associated with the chart. The **Up** and **Down** arrow keys are the best way to do this. When the first symbol is selected, *Previous* becomes *Last*, since the *Up* key will loop back around the bottom of the list. Similarly, when the last symbol is selected, *Next* becomes *First*.

**Jump to Symbol** opens a symbol selection dialog where the desired symbol can be either typed or found in a list of available symbols. The chart is then changed to show that symbol.

**Show / Hide Cursor** toggles the vertical magenta cursor bar on or off. The cursor bar makes it possible to see the data values for any specific bar in the chart.

**Show / Hide Cross-Hair** toggles the + crosshair on or off. The crosshair makes it easier to line up bars in the main chart with the indicator panes, and/or to see specific horizontal levels across the entire chart.

**Show Data** opens a new **List Window** that shows all of the data represented by the chart, including values for each plotted indicator.

**Get Information** is a special feature that is only available if you use **Norgate** as your data source. Selecting this item launches a query of all the fundamental information about a stock that Norgate makes available. This information is written to file in the *Info* sub-directory of the RealTest installation directory. The file is named *XYZ\_INFO.HTML* where *XYZ* is the symbol from the chart. The file is then opened as a new window in your default web browser program.

**Show / Hide Indicator** Pane shows or hides the top indicator pane of the chart window.

**Show / Hide Volume Pane** shows or hides the bottom indicator / volume pane of the chart window.

**Show All Bars** zooms the chart out to the maximum length of time available for the current symbol. Pressing **ESC** after doing this restores the zoom level to whatever it was previously. A very convenient way to quickly find a specific range of dates in a chart is to first *Show All Bars*, then drag the mouse across the desired area to zoom in to that area.

**Black Background** toggles the chart window background color between white and black.

**Candlestick Chart** toggles the price data bar drawing style between OHLC bars and candlesticks.

**Logarithmic Scale** toggles the scaling of the Y (price) axis between arithmetic and logarithmic.

**Split-Adjusted** toggles the price values on the chart between split-adjusted and as-traded (unadjusted).

**Weekly Bars** toggles the chart display between regular daily bars and compressed weekly bars.

**Copy Image To Clipboard** places a bitmap image of the currently displayed chart into the Windows clipboard.

**Save Image As PNG File** prompts for a file path to save the currently displayed chart in PNG image format.

**Edit Chart Formulas** opens a **script window** with the script that contains the **Charts Section** that was last applied to a chart. Most of the time, this will be *charts.rts* unless you've recently applied a different script with a replacement *Charts* section.

**Refresh Chart** forces all the indicators to be recalculated and the contents of the chart to be redrawn.

**Options** opens the chart options dialog box.

## 7.2.2. Graph Menu

---

The *Graph* menu appears in the menu bar whenever a **Stats Graph Window** is active, and can also be accessed by pressing the right mouse button within that window.

Graph	
Show Cursor	.
Show Data	D
Sort	S
<hr/>	
Use Log Scale	L
✓ Use Fixed Height	F
✓ Show Combined as Area	C
Show Moving Averages	M
Show Full Range	ESC
<hr/>	
Edit Graph Formulas	F9
Refresh Graph	F10
<hr/>	
Copy Image to Clipboard	Ctrl+C
Save Image as PNG File...	
<hr/>	
 Show Trade Plots	Ctrl+P
 Show Trade List	Ctrl+T
<hr/>	
Options...	O

Menu items with icons next to them are also accessible using the Tool Bar.

**Show / Hide Cursor** toggles the vertical magenta cursor bar on or off. The cursor bar makes it possible to see the data values for any specific date in the graph.

**Show Data** opens a new **List Window** showing all of the data contained in the current graph.

**Sort** is enabled when the current graph type is a bar graph, and can be used to toggle the bar order (X axis) between *by date* and *by value*. Sorting a bar graph by value makes it easier to see at a glance what the range of values is and how common or rare the extremes are.

**Use Log Scale** toggles the scaling of the price (Y) axis between arithmetic and logarithmic. Log Scale is only available when the graph type is a line graph and all the values in the graph data are greater than zero. This is most typically used for a **Compounded Equity** graph.

**Use Fixed Height** toggles whether to keep the Y axis fixed at the scale required to show all strategies at once when not all strategies are being shown.

**Show Combined as Area** toggles the display of the *combined* line for a multi-strategy graph such as *Equity* as a gray area behind the other lines rather than as its own line.

**Show Moving Averages** toggles the display of a moving average line around the graph line for each strategy. Unlike **Chart Windows**, *Graphs* do not include a mechanism for adding custom indicator lines to the existing stats. Only a simple moving average is currently supported. The length of the average is specified in the Options dialog, and can be increased or decreased using the + and - keys on the numeric keypad.

**Show Full Range** restores the graph to the default of showing the entire range of dates at once. It is possible to zoom in to a subset of the data by dragging the mouse across part of the graph. This item (or the **ESC** key) exits the subset view mode.

**Edit Graph Formulas** opens a **script window** with the script that contains the **Graphs Section** that was last applied to a graph. Most of the time, this will be *graphs.rts* unless you've recently applied a different script with a replacement *Graphs* section.

**Refresh Graph** forces all the data items to be recalculated and the contents of the graph to be redrawn.

**Copy Image To Clipboard** places a bitmap image of the currently displayed graph into the Windows clipboard.

**Save Image As PNG File** prompts for a file path to save the currently displayed graph in PNG image format.

**Show Trade Plots** opens a **Trade Plot Window** for the test behind the graph. This has the same effect as selecting this item in the **Results Window**.

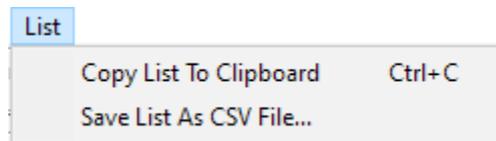
**Show Trade List** opens a **Trade List Window** for the test behind the graph. This has the same effect as selecting this item in the **Results Window**.

**Options** opens the graph options dialog box.

### 7.2.3. List Menu

---

The *List* menu appears in the menu bar whenever a general-purpose **List Window** is active, and can also be accessed by pressing the right mouse button within that window.



The two items on the *List* menu also appear on the menus for all of the more specific list window types.

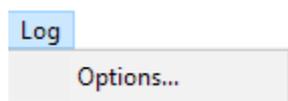
**Copy List To Clipboard** copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

**Save List As CSV File** prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

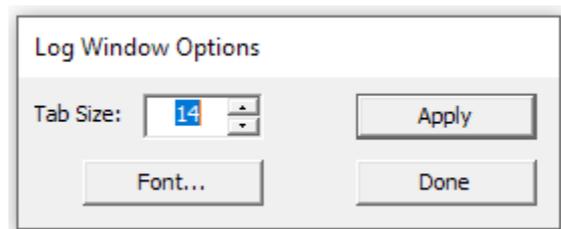
### 7.2.4. Log Menu

---

The *Log* menu appears in the menu bar whenever a **Log Window** is active, and can also be accessed by pressing the right mouse button within that window.



The only item on the Log menu is the choice to open the Log Options dialog box:



The only purpose of the Log Options dialog box is to allow the tab size (how many spaces represent a tab) and the font and size to be specified.

The default tab size of 14 is recommended for best results when viewing a **Test Summary Report**.

### 7.2.5. Plot Menu

---

The *Plot* menu appears in the menu bar whenever a **Plot Window** is active, and can also be accessed by pressing the right mouse button within that window.

Each item on this menu also has a single keystroke associated with it, as shown to the right of the item.

Once you become familiar with these **keyboard shortcuts**, it is rarely necessary to open this menu.

Plot		
<input checked="" type="checkbox"/>	Scatter Plot	S
	Profit Distribution	N
	MAE/MFE Distribution	N
	Cumulative Profit	C
	Monte Carlo Profits	M
	Monte Carlo Drawdowns	D
	Equal Count Bins	B
	Equal Range Bins	R
	Dollars	\$
<input checked="" type="checkbox"/>	Trade Percent	%
	Alloc Percent	^
<input checked="" type="checkbox"/>	Trade Number	T
	Formula	F
	Cross Hair	X
	Linear Regression Line	L
	Log Analysis Stats	A
	Copy Image To Clipboard	Ctrl+C
	Save Image As PNG File...	
	Options...	O

The **first seven items** on this menu cover the seven types of plots available in this window. See **its documentation** for details.

**Dollars, Trade Percent** or **Alloc Percent** specify the unit to display in the Y axis.

**Trade Number** or **Formula** specify the unit to display in the X axis, and therefore the sort order of the data in the plot.

**Cross Hair** toggles the + crosshair on or off.

**Linear Regression Line** toggles the display of the linear regression of the data in the plot.

**Log Analysis Stats** toggles the logging mode for certain plot types.

**Copy Image To Clipboard** places a bitmap image of the currently displayed plot into the Windows clipboard.

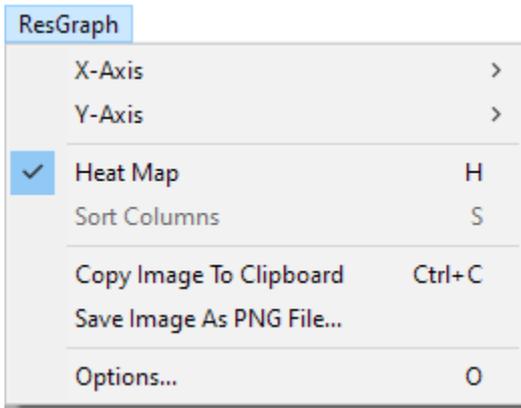
**Save Image As PNG File** prompts for a file path to save the currently displayed plot in PNG image format.

**Options** opens the plot options dialog box.

## 7.2.6. ResGraph Menu

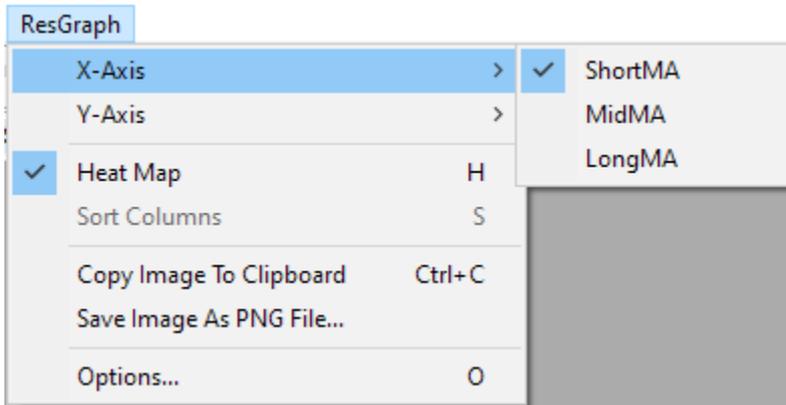
---

The *ResGraph* menu appears in the menu bar whenever a **Results Graph Window** is active, and can also be accessed by pressing the right mouse button within that window.

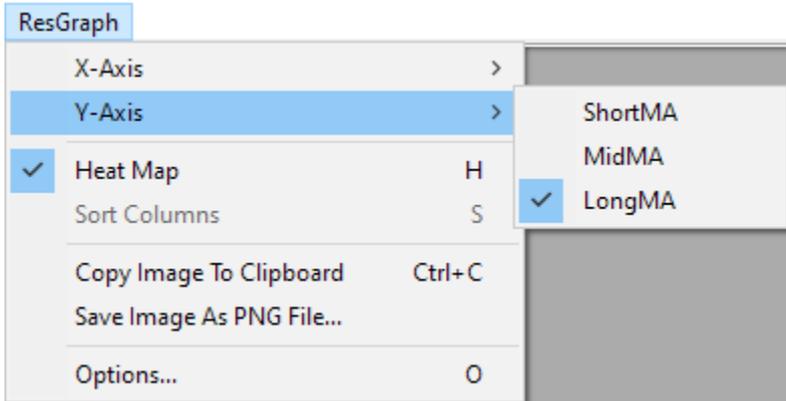


This menu has several sub-menus, as indicated by the > signs.

**X-Axis** opens a sub-menu containing a list of all the parameters that were included in the optimization. The selected parameter becomes the X-Axis of the graph.



**Y-Axis** opens a sub-menu containing a list of all the parameters that were included in the optimization. The selected parameter becomes the Y-Axis of the graph. Note that this sub-menu will only be accessible when the graph is shown as a Heat Map.



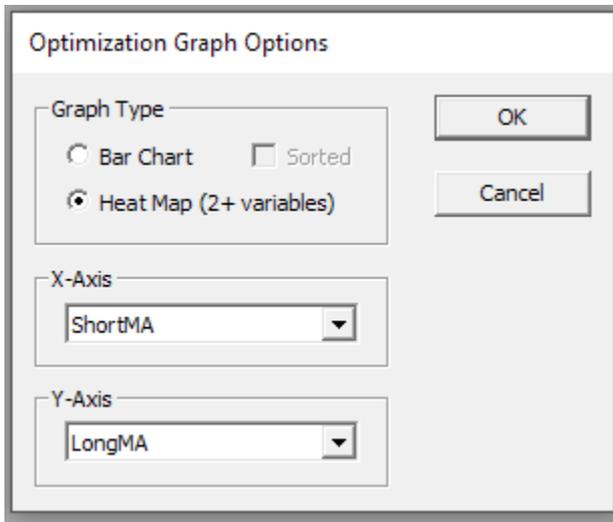
**Heat Map** toggles the optimization graph between a bar graph and a heat map. The latter is only available when there are two or more parameters.

**Sort Columns** (bar chart only) toggles the bar order (X axis) between *test number* and *value*. Sorting an optimization graph by value makes it easier to see at a glance what the range of values is and how common or rare the extremes are. This item is only available when *Heat Map* is not selected.

**Copy Image To Clipboard** places a bitmap image of the currently displayed graph or heat map into the Windows clipboard.

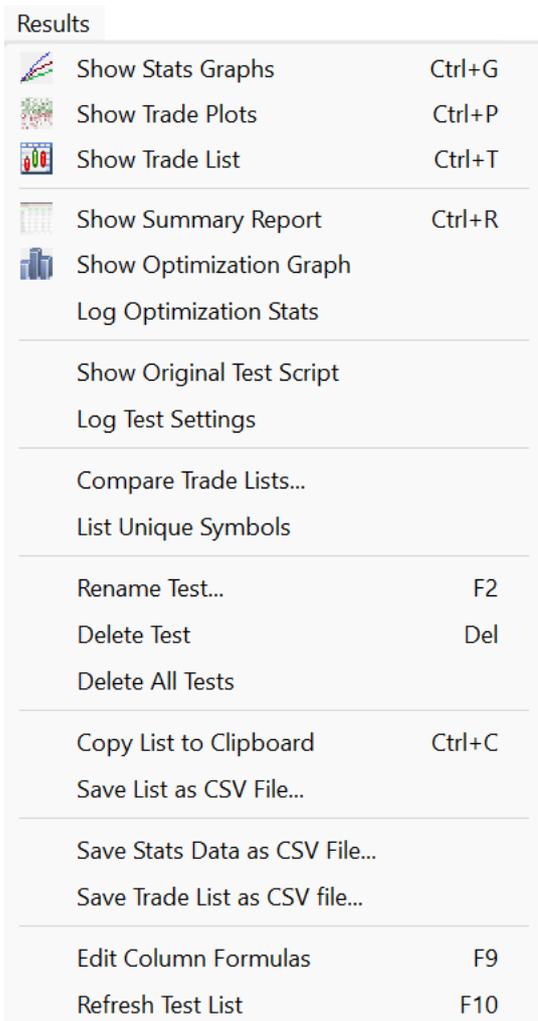
**Save Image As PNG File** prompts for a file path to save the currently displayed graph or heat map in PNG image format.

**Options** opens the optimization graph options dialog box, which is an entirely redundant way to access the same set of choices that are on the menu.



## 7.2.7. Results Menu

The *Results* menu appears in the menu bar whenever a **Results Window** is active, and can also be accessed by pressing the right mouse button within that window.



Menu items with icons next to them are also accessible using the Tool Bar.

**Show Stats Graph** opens a **Daily Stats Graph Window** for the selected test. This can also be done by double-clicking on the test row.

**Show Trade Plots** opens a **Trade Plot Window** for the selected test.

**Show Trade List** opens a **Trade List Window** for the selected test.

**Show Summary Report** generates a **Test Summary Report** for the selected test.

**Show Optimization Graph** opens an **Optimization Graph Window** for all the tests in this *Results Window*.

**Log Optimization Stats** opens a **Log Window** and writes various summary stats for all the tests in this *Results Window*.

**Show Original Test Script** opens a read-only **Script Window** allowing you to view the exact script that was used to run the selected test.

**Log Test Settings** opens a Log Window that shows the key settings that were used to run the selected test.

**Compare Trade Lists** opens a dialog box to select two Test+Strategy pairs, then opens a **Trade Comparison Window**.

**List Unique Symbols** opens a **Log Window** and writes a sorted list of each symbol that appears at least once in the selected test.

**Rename Test** allows the text in the *Name* column of the selected test to be modified.

**Delete Test** deletes the selected test from the *Results Window*. This cannot be undone.

**Delete All Tests** clears the current *Results Window*.

**Copy List To Clipboard** copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

**Save List As CSV File** prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

**Save Stats Data as CSV File** prompts for a file path and name, then all formulas in the Graphs section are evaluated for the combined stats series for each date of the test and written in CSV columns as raw values (format codes are ignored).

**Save Trade List as CSV File** prompts for a file path and name, then all standard and custom trade items for every trade in the test are written in CSV columns as raw values (format codes are ignored).

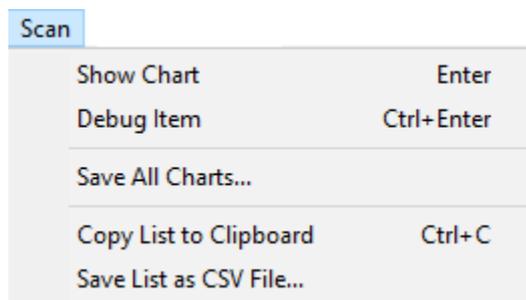
**Edit Column Formulas** opens a **script window** with the script that contains the **Results Section** that was last applied to a *Results Window*. Most of the time, this will be *results.rts* unless you've recently applied a different script with a replacement *Results* section.

**Refresh Test List** forces all the columns to be recalculated and the contents of the window to be redrawn.

## 7.2.8. Scan Menu

---

The *Scan* menu appears in the menu bar whenever a **Scan Window** is active, and can also be accessed by pressing the right mouse button within that window.



**Show Chart** opens a chart of the symbol in the selected row, with the date from the selected row as the rightmost bar. Use the **up** and **down** arrow keys on this chart to quickly and easily cycle through the charts of all the items in the list.

**Debug Item** opens the **Debug Panel** and sets its date and symbol fields to match the selected scan

row.

**Save All Charts** prompts for a folder path, then creates a chart for every row of the scan (symbol\_date.png) and saves it to that folder. Existing contents are not deleted first, but files with identical names will be replaced without asking. Saved charts have the same width and height and other display options as the most recently viewed chart window. This can also be done automatically by adding **SaveChartsTo** to the **ScanSettings** in a script.

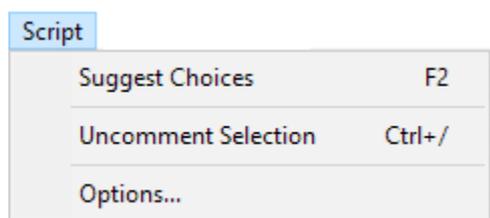
**Copy List To Clipboard** copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

**Save List As CSV File** prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

## 7.2.9. Script Menu

---

The *Script* menu appears in the menu bar whenever a **Script Window** is active, and can also be accessed by pressing the right mouse button within that window.



Most of the editing functions used in a *Script Window* can be found on the **Edit Menu**.

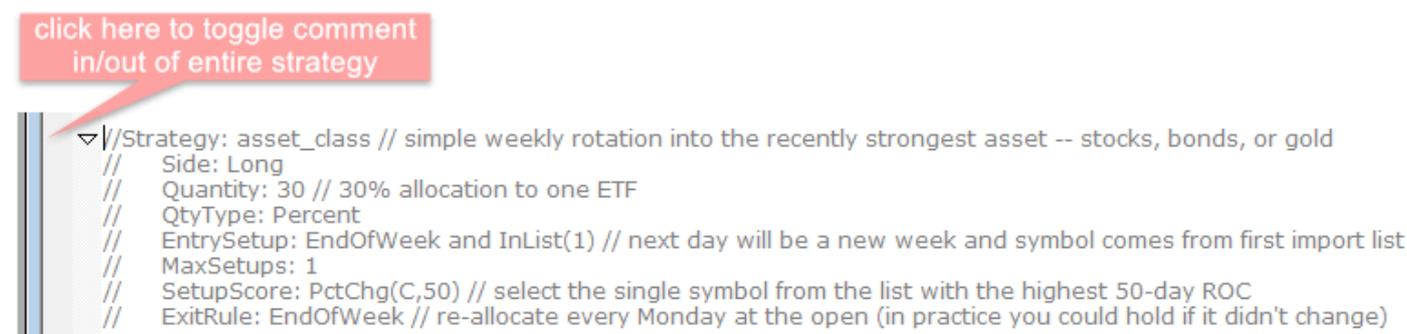
When the right mouse button is pressed in a script window, the popup version of this menu includes both Edit and Script menu items.

**Suggest Choices** invokes the auto-complete feature of the editor as if you had not typed anything yet. This pops up a list of every syntax element that is allowed to be used at the point in the script where the cursor is currently located.

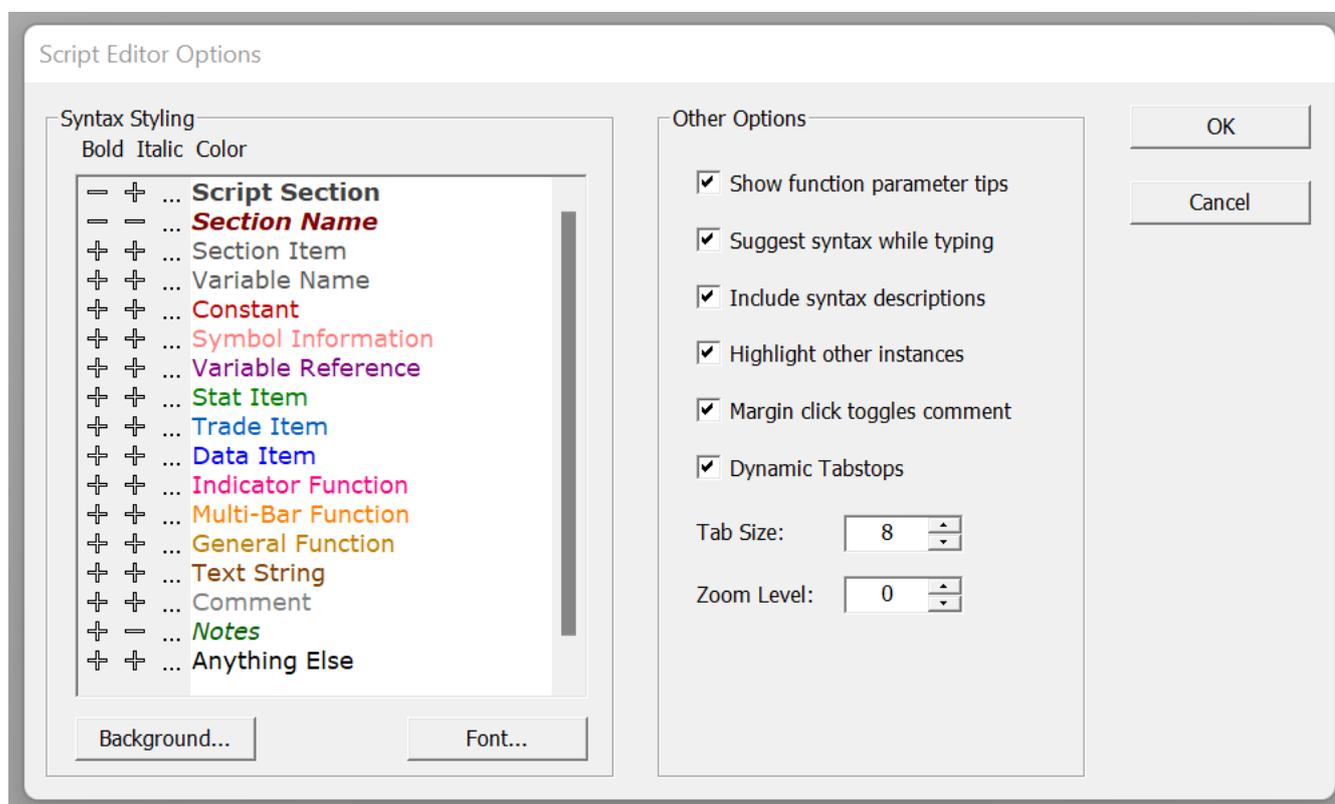
**Comment / Uncomment Selection** toggles the commented state of the current line or selected set of lines. This is done by either inserting or removing // comment marks from the start of each line. If the cursor is on a **Script Section** definition line when this item is selected, the commenting of the entire section is toggled.

Note that the shortcut key for this menu item is Ctrl+/, which is easy to remember since // is the line comment syntax.

Another convenient shortcut to this feature is to click the mouse in the leftmost vertical margin of the script window:



**Options** opens the Script Editor Options dialog box:



The section in the left half of this dialog facilitates the choice of which color to use for each type of **syntax element** that can appear in a script.

To modify a color, click on the ellipses just to the left of an element type name.

Click on the + or - signs to the left of the ellipses to modify the bold and/or italic look for that element.

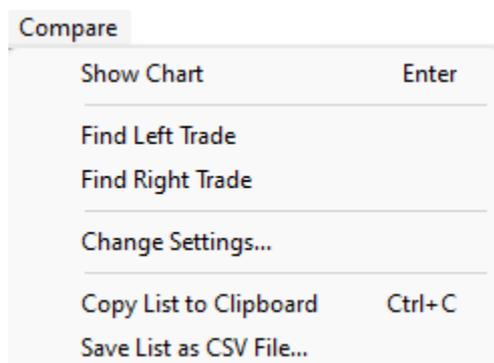
Click on *Background* to change the background color or *Font* to change the typeface or size.

Other options include:

- whether to auto-complete the parameter list for a function when the open parenthesis is typed
- whether to offer auto-complete in general (the entire feature can be disabled here)
- whether to show syntax element descriptions within the suggestion list during auto-complete (these are not inserted in the script, just shown in the popup)
- whether to auto-highlight any other copies of the currently selected word in the script
- whether clicking in the leftmost margin of the script window will toggle the comment state of that line or instead select the entire line
- an option to dynamically create an optimal set of tabstop locations based on the text of the entire script
- the fixed tab size when not using dynamic tabstops
- the current font zoom level (can also be modified using the Ctrl+mouse wheel)

## 7.2.10. Compare Menu

The *Compare* menu appears in the menu bar whenever a **Trade Comparison Window** is active, and can also be accessed by pressing the right mouse button within that window.



**Show Chart** opens a **Chart** of the symbol involved in the selected trade, showing the entry and exit bars and price levels on the chart. If a row contains two trades, both of their entries and exits are shown. Use the **up** and **down** arrow keys on this chart to quickly and easily cycle through the charts of all the trades in the list.

**Find Left Trade** opens a **Trade List Window** and highlights the specific trade shown on the left side of the selected row, so that you can see more details about it if needed. **Find Right Trade** does the same for the trade shown on the right side of the selected row.

**Change Settings** reopens the dialog box that was used to open the *Trade Comparison* and allows you to update it with different specifications.

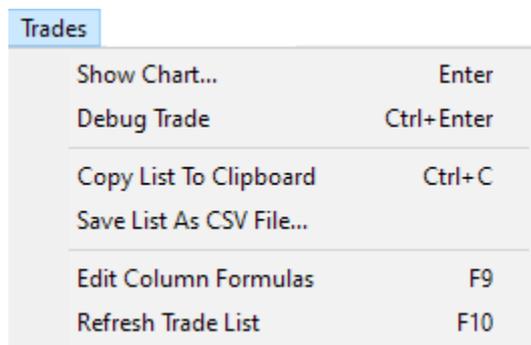
**Copy List To Clipboard** copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

**Save List As CSV File** prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

## 7.2.11. Trades Menu

---

The *Trades* menu appears in the menu bar whenever a **Trade List Window** is active, and can also be accessed by pressing the right mouse button within that window.



**Show Chart** opens a **Chart** of the symbol involved in the selected trade, showing the entry and exit bars and price levels on the chart. Use the **up** and **down** arrow keys on this chart to quickly and easily cycle through the charts of all the trades in the list.

**Debug Trade** opens the **Debug Panel** and sets its date, symbol, test and strategy fields to match the selected trade.

**Copy List To Clipboard** copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

**Save List As CSV File** prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

**Edit Column Formulas** opens a **Script Window** with the script that contains the **Trades Section** that was last applied to a *Trade List Window*. Most of the time, this will be *trades.rts* unless you've recently applied a different script with a replacement *Trades* section.

**Refresh Trade List** forces all of the columns to be recalculated and the contents of the window to be

redrawn.

## 7.3. Bars and Panels

---

In addition to the menu bar, which is always visible, RealTest has four other bars or panels which can be either shown or hidden.

### 7.3.1. Tool Bar

---

The primary functions that you will use in your RealTest workflow are the following script run commands:

- Check and Apply (F4)
- Run Test (F5)
- Run Optimization (F6)
- Run Import (F7)
- Run Scan (F8)
- Run Orders (Ctrl+F5)
- Stop (Ctrl+Break)

These commands can be accessed in multiple ways:

- From the **Tool Bar**:



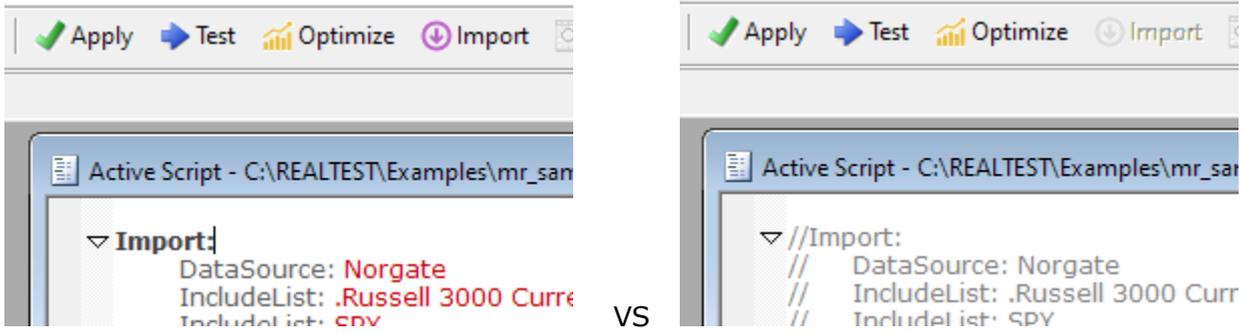
- From the **Run Menu**

Run	Script	Window	Help
	Check and Apply		F4
	Run Test		F5
	Run Optimization...		F6
	Run Import		F7
	Run Scan		F8
	Run Orders		Ctrl+F5
	Stop		Ctrl+Break

- By using function keys as shown above

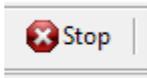
RealTest automatically enables or disables each specific command in the Run Menu and on the Tool Bar depending on the contents of the active script.

For example, if the script does not contain an "Import" section, the Import button will be disabled.

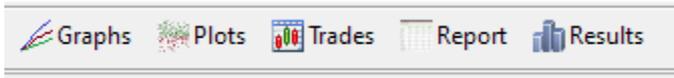


The run commands for the active script remain accessible even when a non-script window (results, graph, chart, etc.) is in the foreground.

A running script can be stopped at any time, such as when you realize you ran the wrong import, don't like the test results so far, etc. by clicking the stop button or pressing Ctrl+Break.

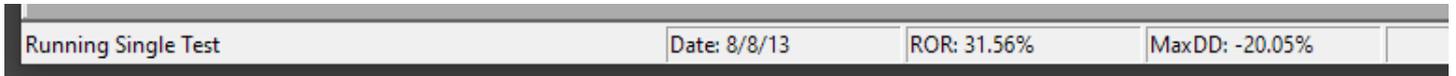


After a test is run, the Tool Bar buttons that invoke **Results Analysis** features are enabled.



### 7.3.2. Status Bar

During the running of a script, the status bar at the bottom of the main window displays dynamically updated information about the progress of the running task.



In many cases, scripts only take a few seconds to run, so you might not notice this, but for longer tasks, it is useful to watch the progress.

If the script is a test, some of the summary statistics are displayed in the fields on the right side of the status bar while the test is running.

Which stats are displayed can be customized using the results script:

```

Active Script - C:\RealTest\Results.rts
Notes: this is the default set of columns for Results windows {...}
Results: // column      {format}      formula
Periods:      {#}          S.Number - S.First + 1 // number of periods from first trac
// use S.Number alone if you prefer "Periods" (and stats) t
NetProfit:    {$0}        S.Equity - S.StartEquity - S.CashInOut
// Dividends:  {$0}        Sum(S.Dividends, Periods)
// Interest:   {$0}        Sum(S.Interest, Periods)
// Cash:       {$0}        S.CashInOut // already cumulative
// NetFX:      {$0}        Sum(S.NetFx, Periods)
_years:      Days(S.Date[Periods-1], S.Date) / 365.25
_CAR:        S.TWEQ^(1/_years) - 1
_AAR:        (((S.Equity - S.CashInOut) / S.StartEquity) - 1) / _years
Comp:        {~}         S.Compounded // shows whether stats were compounded
// Mark:      {~}         Combined(S.MarkedToMarket) // shows whether equity wi
ROR:         {%2|}       iif(Comp, _CAR, _AAR)
MaxDD:       {%2|}       -S.MaxDDPct

```

The vertical bar character "|" in the formatting codes indicates that a results item should be displayed dynamically in the status bar while a test is running. By default, the items shown as scripts run are: ROR, MaxDD, Exits, Expectancy, and ProfitFactor.

### 7.3.3. Settings Panel

The **Settings Panel** is used to specify all the settings needed to run tests or scans.

All these settings (except the active script name) can **optionally be specified** within a script.

This gives you the choice to either use this panel interface, a pure script-based approach, or a mixture of the two.

This panel can be hidden or shown using the **View Menu** or by pressing the F11 key.

The **Settings Panel** includes the following items:

❖ **Data File**

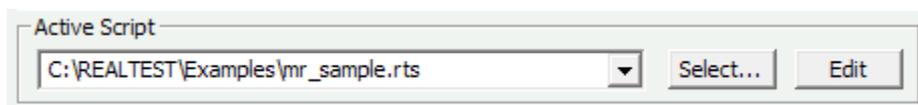


Specifies a previously **imported .RTD file** to use in a test or scan.

This file is automatically loaded into memory (if not already there) when a script is run.

Use the load button to load it manually.

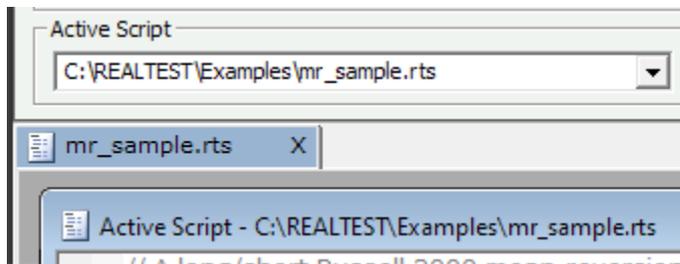
❖ **Active Script**



When any of the **script run commands** is chosen, it is applied to the currently active script window.

This will always be the script which was most recently opened or edited.

It is designated as "Active Script" in its title and its path is automatically shown in the "Active Script" field on the settings panel (if visible).



It is also possible to run any script without first opening it in a window by selecting it on the settings panel.

A script selected on the settings panel will remain active until a different script is selected or a script window is opened.

#### ❖ Test Name

A place to type a brief note that will be displayed in the **results window** row for the test after it is run.

#### ❖ Account Size

Specifies the amount of money to use in the simulated account when running tests.

#### ❖ Date Range

Specifies the range of dates to use for the next test or scan.

#### ❖ Bar Size

Specifies the default data time period to use when running a script or using the debug panel. Currently supported bar sizes are Daily, Weekly, and Monthly. Weekly and Monthly bars are derived from daily bars, so it is not necessary (nor supported) to import weekly or monthly data.

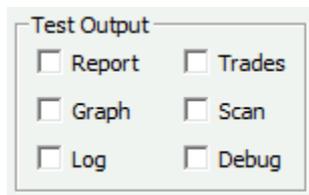
#### ❖ Keep Trades

Specifies the categories of trade records to keep in each test result record (each row of the Results window).

Options are:

- **Strategy** - include normal strategy trades
- **Benchmark** - include benchmark strategy trades
- **Skipped** - include skipped trades (from each of the above categories if selected)

#### ❖ Test Output



Specifies which type of additional output to create when a test is run besides the usual results data.

Some of these (Report, Trades, Graph) can also be run manually at any time by selecting a row in **Results** and then clicking the corresponding toolbar button.

The others (Scan, Log, Debug) must be selected before a test is run in order to be generated by that test.

Available choices are:

- **Report** - generates the per-strategy and per-month summary report at the end of the test
- **Trades** - displays the round-trip trade list at the end of the test
- **Graph** - opens a stats graph at the start of the test and updates it dynamically as the test runs
- **Scan** - runs a **TestScan** simultaneously with the test, allowing fully customizable output of test-related data
- **Log** - generates a detailed transaction and position log and displays it at the end of the test
- **Debug** - opens the **debug panel** before closing end-of-test positions to allow full examination of the test context on the last date, and whether to show output from **DebugEntry**, **DebugExit** and **DebugTargetStop** statements (if any) in the script

## 7.3.4. Debug Panel

---

The **Debug Panel** facilitates detailed examination of **data** and/or **test stats**, either for interactive analysis or to help with formula construction.

It contains a mini **formula** editor which is used to produce immediate output to a **log window**, with optional display of evaluation detail.

It also allows the data for any symbol (including **calculated columns**) or the underlying stats from any test to be viewed in tabular format.

When accessed at the end of a test run with *Debug* selected for **TestOutput**, the full strategy formula context is also available for examination.

These capabilities make it possible to fully understand what's happening "under the hood" of a formula or backtest.

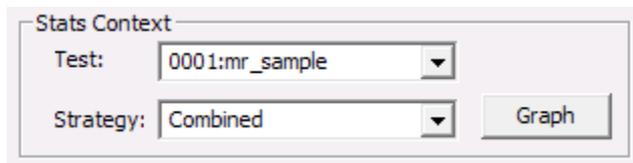
The **Debug Panel** includes the following items:

### ❖ Data Context



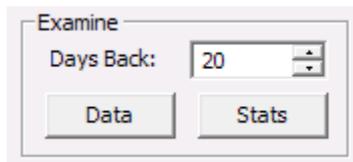
Specifies the symbol and most recent date to use for either showing a **bar chart**, showing underlying bar data, or evaluating a formula with data elements.

### ❖ Stats Context



Specifies the test (from the set of tests in the currently active **Results window**) and strategy to use for either showing a stats graph, showing the underlying stats data, or evaluating a formula with stats elements.

❖ **Examine**



Specifies how many days back from the Data Context date to show when either the Data or Stats button is pressed.

❖ **Positions**



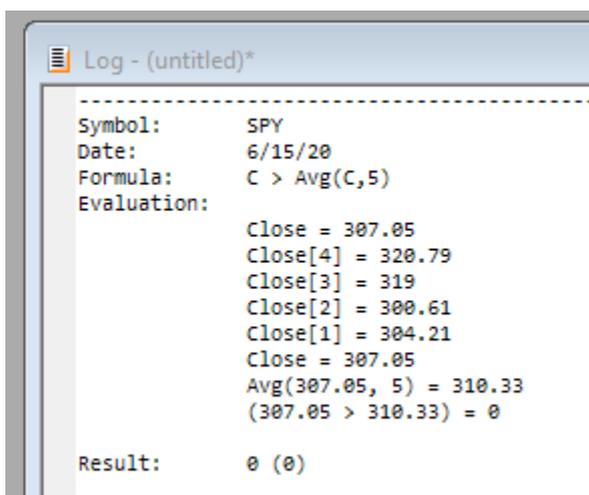
Logs the list of open positions at the end of the last test that was run.

❖ **Formula**



This mini script editor lets you type any formula for immediate evaluation. If Show Evaluation is checked, every step of the evaluation is documented.

Log output after pressing "Evaluate":



List output after pressing "Data" (not all columns shown):

Data - SPY - 20 days back from 6/15/20								
Date	Open	High	Low	Close	Volume	Split	ATRx	EMAx
6/15/20	298.02	308.28	296.74	307.05	135,782,720	1	8.92	308.70
6/12/20	308.24	309.08	298.6	304.21	194,678,880	1	8.26	309.53
6/11/20	311.46	312.15	300.01	300.61	209,243,552	1	7.71	312.19
6/10/20	321.42	322.39	318.221	319	95,000,768	1	4.89	317.98
6/9/20	320.3	323.285	319.36	320.79	77,479,232	1	5.07	317.47
6/8/20	320.22	323.41	319.63	323.2	73,641,216	1	5.36	315.81
6/5/20	317.23	321.275	317.16	319.34	150,524,672	1	5.68	312.11
6/4/20	311.11	313	309.08	311.36	75,794,360	1	4.62	308.50
6/3/20	310.24	313.22	309.94	312.18	92,567,576	1	4.79	307.07
6/2/20	306.55	308.13	305.1	308.08	74,267,160	1	4.71	304.52
6/1/20	303.62	306.205	303.06	305.55	56,779,836	1	5.13	302.74
5/29/20	302.46	304.96	299.47	304.32	119,265,704	1	5.62	301.33
5/28/20	304.65	306.84	302.24	302.97	90,767,808	1	5.66	299.84
5/27/20	302.12	303.57	296.87	303.53	104,817,448	1	5.92	298.27
5/26/20	301.93	302.19	295.465	299.08	88,951,440	1	5.73	295.64
5/22/20	294.57	295.63	293.22	295.44	63,958,200	1	5.47	293.92
5/21/20	296.79	297.67	293.689	294.88	78,293,928	1	6.24	293.16
5/20/20	295.82	297.87	295.57	296.93	85,861,688	1	6.80	292.30
5/19/20	294.35	296.205	291.95	291.97	95,189,312	1	7.03	289.99
5/18/20	293.05	296.75	292.7	295	120,320,232	1	7.72	288.99
5/15/20	282.37	286.33	281.34	286.28	111,146,272	1	7.03	285.99

There are more columns in the above list. All user-defined data column values are included after the standard bar data.

List output after pressing "Stats" (not all columns shown):

Stats - Test 1 (mr_sample) - Strategy @Combined											
Date	Equity	Drawdown	DDBars	M2M	MAE	MFE	Setups	Entries	Exits	Positions	Exposure
6/15/20	\$673,300	-1.30%	1	-1.32%	0	0	14	2	0	2	-20.21%
6/12/20	\$682,167	0	0	0	-2.43%	5.92%	210	0	10	0	0
6/11/20	\$656,805	0	0	-0.83%	-3.15%	5.62%	53	11	3	10	99.10%
6/10/20	\$650,336	0	0	-0.25%	-1.00%	1.26%	19	1	2	2	-19.63%
6/9/20	\$646,537	-0.11%	8	-0.38%	-2.87%	1.91%	97	1	6	3	-9.63%
6/8/20	\$631,827	-2.38%	7	-2.49%	-7.29%	2.12%	89	1	5	8	-59.79%
6/5/20	\$636,408	-1.67%	6	-1.72%	-7.97%	2.49%	84	7	3	12	-79.40%
6/4/20	\$633,258	-2.16%	5	-0.50%	-2.84%	0.91%	107	5	3	8	-59.87%
6/3/20	\$630,778	-2.54%	4	-1.40%	-3.25%	0.61%	48	2	4	6	-60.12%
6/2/20	\$633,481	-2.12%	3	-1.25%	-2.36%	2.26%	67	1	2	8	-79.94%
6/1/20	\$641,963	-0.81%	2	-0.11%	-1.21%	2.36%	65	6	3	9	-88.78%
5/29/20	\$640,686	-1.01%	1	0.45%	0	2.14%	24	3	0	6	-19.55%
5/28/20	\$647,221	0	0	1.46%	-4.08%	7.38%	108	1	14	3	-9.39%
5/27/20	\$610,458	-2.75%	411	-0.98%	-3.40%	2.08%	62	13	4	16	-18.80%
5/26/20	\$609,277	-2.94%	410	0.79%	-1.48%	3.38%	45	7	4	7	-48.65%
5/22/20	\$594,543	-5.29%	409	-0.12%	-4.22%	1.38%	35	2	4	4	-39.64%
5/21/20	\$592,332	-5.64%	408	-2.14%	-4.13%	1.71%	46	2	5	6	-39.66%
5/20/20	\$592,586	-5.60%	407	-1.98%	-5.63%	2.05%	40	3	2	9	-69.13%
5/19/20	\$583,814	-7.00%	406	-2.09%	-3.84%	1.25%	79	2	2	8	-80.22%
5/18/20	\$595,765	-5.09%	405	-0.08%	-1.89%	1.28%	54	9	4	8	-78.28%

There are many more columns in this list – all custom and built-in stats values are included.

These list windows also make it easy to save the underlying data to CSV format or copy/paste directly to Excel for further analysis.

Log output after running a test with "End Of test" set to "Debug" and then pressing "Positions":

-----  
positions open for mr\_sample\_debug run with end date 11/13/20  
-----

Strategy	Symbol	DateIn	TimeIn	QtyIn	PriceIn
mr_short	PLUG	11/12/20	intraday	1058	24.04
mr_short	APPN	11/13/20	open	260	99.51
mr_short	FLGT	11/13/20	open	637	40.65
mr_short	CYRX	11/13/20	intraday	415	62.32
mr_short	GRWG	11/13/20	intraday	1005	25.76
mr_short	LOVE	11/13/20	intraday	920	28.13
mr_short	XPEL	11/13/20	intraday	756	34.26

## 7.4. Text Editor Windows

---

RealTest integrates the wonderful open-source Scintilla text editor for use in child windows of the following types.

Scintilla Copyright 1998-2002 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

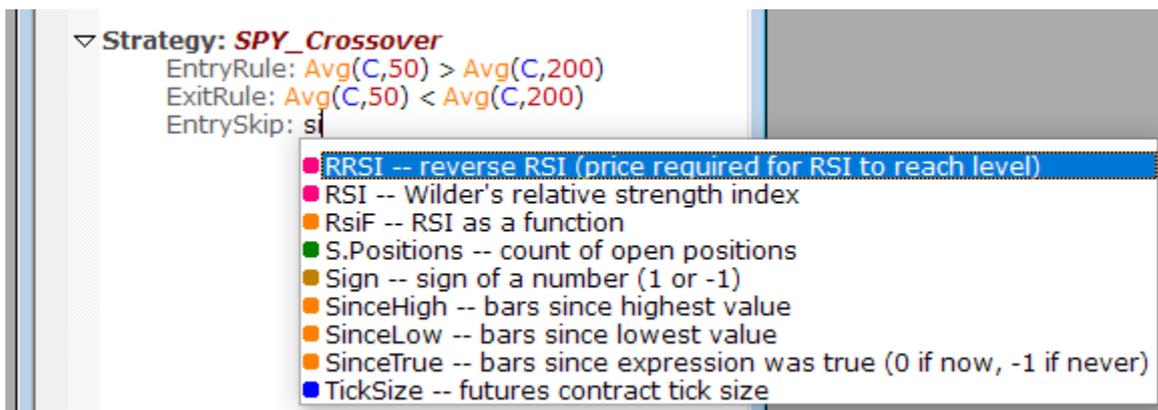
Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

### 7.4.1. Script Windows

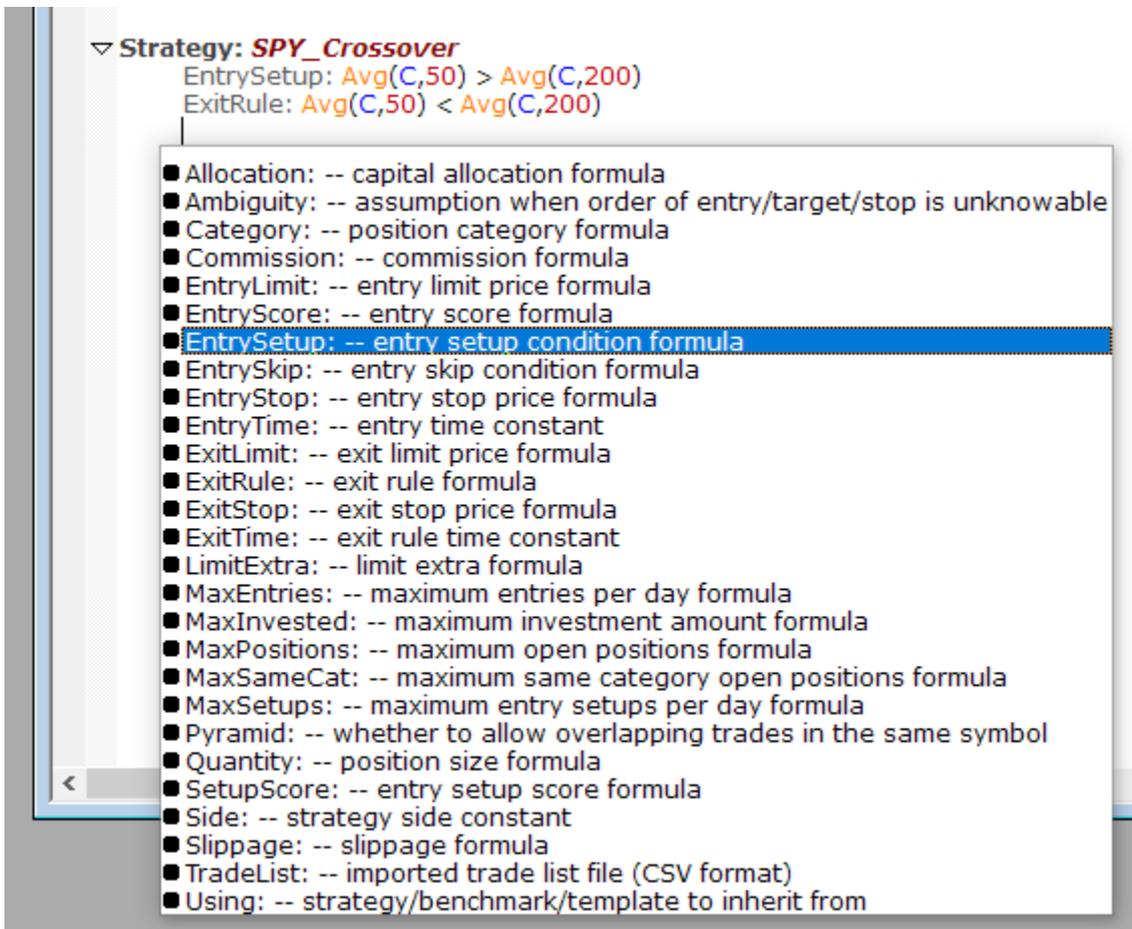
---

Script windows take full advantage of the language-sensitivity features of Scintilla, such as color-coded syntax, popup help, auto-completion, function parameter prompting, parentheses balancing, multi-level undo/redo, smart tabbing, block comment in/out, etc.



The popup above shows all available syntax containing "si", since that is what was typed so far.

You can also press F2 in a script window to see all available syntax options for where the cursor is located.



The cursor was indented one tabstop, under "ExitRule" within the Strategy section, so the list shows all available strategy elements.

You can also position the cursor within any syntax element and press F1. This will open the RealTest Help window with the relevant help for that syntax element.

See the **Script Menu** documentation for more about what can be done in a Script Window.

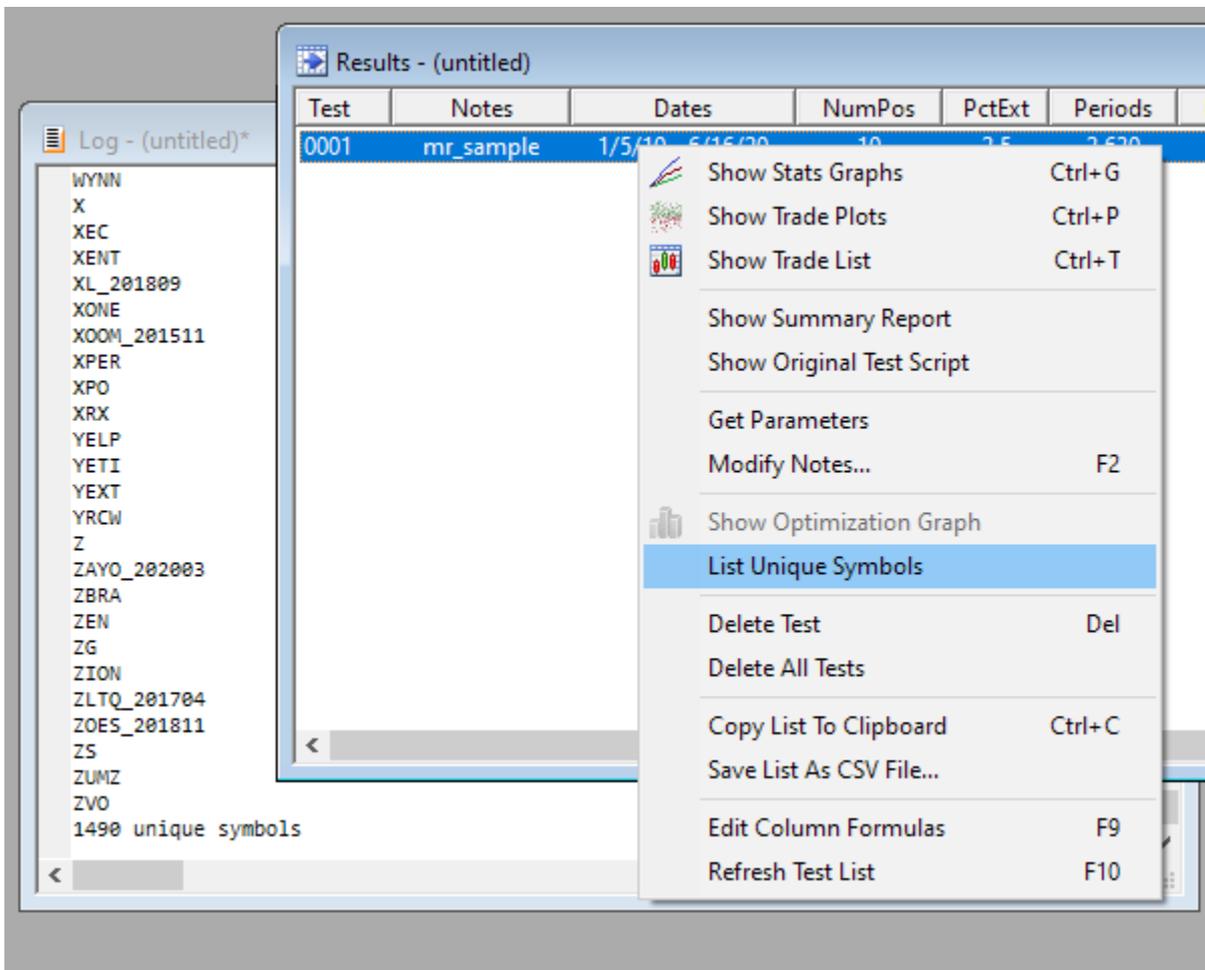
## 7.4.2. Log Windows

---

Log windows serve to display some types of optional output from a test, or output from the debug panel. You can also open any text file as a log window, edit it, and re-save it, such as you might do in the Windows Notepad app. You might also use a log window to keep notes about your research process.

RealTest includes many features that use log windows for their output, such as the **Test Details Log** and **Tomorrow's Orders** list.

Another feature that uses a log window is the ability to list all the symbols that were used in a test result:



Another example is the **Debug Panel** interactive formula evaluation output. There are many others as well.

## 7.5. List Windows

List windows display tabular data in rows and columns.

The Results, Trades and Scan windows are the most commonly used windows of this type.

Data can be sorted by any column by clicking on its header. Clicking again reverses the sort order.

Test	Notes	Dates	ShortMA	LongMA	Periods	NetProfit	ROR ▼	MaxDD
0030	Sample2	2/1/93 - 10/27/20	60	250	6,987	\$1,017,939	9.10%	-34.05%
0035	Sample2	2/1/93 - 10/27/20	30	300	6,987	\$972,095	8.93%	-34.05%
0036	Sample2	2/1/93 - 10/27/20	40	300	6,987	\$957,059	8.88%	-34.12%
0021	Sample2	2/1/93 - 10/27/20	50	200	6,987	\$937,972	8.81%	-34.12%
0037	Sample2	2/1/93 - 10/27/20	50	300	6,987	\$918,500	8.73%	-34.10%
0040	Sample2	2/1/93 - 10/27/20	80	300	6,987	\$914,576	8.72%	-34.09%
0022	Sample2	2/1/93 - 10/27/20	60	200	6,987	\$895,938	8.64%	-34.15%
0028	Sample2	2/1/93 - 10/27/20	40	250	6,987	\$892,999	8.63%	-34.13%
0025	Sample2	2/1/93 - 10/27/20	10	250	6,987	\$886,870	8.61%	-19.01%

Shift-click on another column to use it as a secondary sort when values in the first sort column are the same. Shift-clicking again reverses the secondary sort direction.

Test	Notes	Dates	ShortMA ▲	LongMA ▼	Periods	NetProfit	ROR	MaxDD
0033	Sample2	2/1/93 - 10/27/20	10	300	6,987	\$756,085	8.05%	-23.52%
0025	Sample2	2/1/93 - 10/27/20	10	250	6,987	\$886,870	8.61%	-19.01%
0017	Sample2	2/1/93 - 10/27/20	10	200	6,987	\$801,987	8.26%	-19.03%
0009	Sample2	2/1/93 - 10/27/20	10	150	6,987	\$593,201	7.23%	-27.15%
0001	Sample2	2/1/93 - 10/27/20	10	100	6,987	\$667,718	7.63%	-33.91%
0034	Sample2	2/1/93 - 10/27/20	20	300	6,987	\$841,845	8.42%	-32.96%
0026	Sample2	2/1/93 - 10/27/20	20	250	6,987	\$816,128	8.32%	-30.49%
0018	Sample2	2/1/93 - 10/27/20	20	200	6,987	\$790,787	8.21%	-30.97%

Sort priority and direction is shown using colored triangles, in "rainbow order" -- red, yellow, green, blue.

List window contents can be copied to the clipboard in tab-delimited format for easy pasting to Excel with columns preserved, and/or saved to disk in CSV format.

In Results, Scan and Trade list windows you can "drill down" to the detail behind any row by selecting the row and then using the context menu. Double-clicking or pressing the Enter key opens the most commonly used type of detailed data, such as a chart for a trade list or a stats graph for a test.

Test	Notes	Dates	MA1	MA2	Periods	NetProfit
0024	Sample2	2/1/93 - 6/16/20	6	30	6,894	\$331,416
0023	Sample2	2/1/93 - 6/...				
0015	Sample2	2/1/93 - 6/...				
0019	Sample2	2/1/93 - 6/...				
0076	Sample2	2/1/93 - 6/...				
0016	Sample2	2/1/93 - 6/...				
0052	Sample2	2/1/93 - 6/...				
0018	Sample2	2/1/93 - 6/...				
0074	Sample2	2/1/93 - 6/...				
0077	Sample2	2/1/93 - 6/...				
0098	Sample2	2/1/93 - 6/...				
0020	Sample2	2/1/93 - 6/...				
0075	Sample2	2/1/93 - 6/...				
0073	Sample2	2/1/93 - 6/...				
0070	Sample2	2/1/93 - 6/...				
0087	Sample2	2/1/93 - 6/...				
0097	Sample2	2/1/93 - 6/...				
0069	Sample2	2/1/93 - 6/...				
0086	Sample2	2/1/93 - 6/...				

After a graph or chart is opened in this way, it remains linked to the list that it came from. By using the UP and DOWN arrow keys, you can flip through the charts of all trades in a test or symbols in a scan, or the equity curves of all tests in a set of results.



Closing a list window will also close all the graphs or charts that were associated with it.

## 7.5.1. Results Windows

Much of the usefulness of RealTest comes from reviewing test results, getting new ideas from this review, modifying the strategy definition, and repeating this process.

After a test is run, a new row is added to a results window showing the summary stats from the test and providing a gateway to every available detail behind those stats.

Test	Name	Dates	Periods	NetProfit	ROR	ShortMA	LongMA
0040	Sample2	2/1/93 - 9/15/20	6,877	\$917,193	8.87%	80	300
0039	Sample2	2/1/93 - 9/15/20	6,887	\$856,933	8.62%	70	300
0038	Sample2	2/1/93 - 9/15/20	6,897	\$802,994	8.37%	60	300
0037	Sample2	2/1/93 - 9/15/20	6,907	\$921,662	8.85%	50	300
0036	Sample2	2/1/93 - 9/15/20	6,917	\$962,425	8.99%	40	300
0035	Sample2	2/1/93 - 9/15/20	6,927	\$974,651	9.02%	30	300

other columns here...

parameter values at end of row

Results analysis begins with simply looking at the columns in the results window.

The first three columns are always the test number, test notes (specified either in the **Settings Panel** or the script), and date range (ditto).

After that come all the user-defined columns as described below.

Finally, if there were **Parameter** values in any test shown, those are displayed as the rightmost columns, as shown above.

By default, the results column contents are defined in the script file RESULTS.RTS located in the same directory as the RealTest program:

▼ **Notes:** this is the default set of columns for Results windows  
edit this file then press F4 or click Apply to change your results columns

items that begin with an underscore are not displayed -- they're used in lieu of repeating an expression  
items with | in the format code appear in the status bar while tests are running

you can also add a Results section to any script  
this default one is only used when there is not a custom one

additional columns that you might want are commented out below

```
▼ Results: // column      {format}      formula
Periods:      {#}          S.Number - S.First + 1 // stats begin when first position is entered
NetProfit:    {$0}         S.Equity - S.StartEquity - S.CashInOut

//           Dividends:    {$0}          Sum(S.Divs, Periods)
//           NetFX:        {$0}          Sum(S.NetFx, Periods)

           _PPY:          Periods / S.BPY
           _CAR:          S.TWEQ^(1/_PPY) - 1
           _AAR:          (((S.Equity - S.CashInOut) / S.StartEquity) - 1) / _PPY
Comp:         {~}         Combined(S.Compounded) // shows whether stats were compounded

//           Mark:         {~}          Combined(S.MarkedToMarket) // shows whether equity was marked to market

ROR:          {%2|}       iif(Comp, _CAR, _AAR)
MaxDD:        {%2|}       -S.MaxDDPct

//           RoMdd:        {#2}         ROR / -MAXDD // return over maxdd aka Mar Ratio, Calmar Ratio
//           RoAdd:        {#2}         ROR / Avg(S.DDPct, Periods) // return over avgdd
//           UI:           {#}          Sqr(SumSq(100*S.DDPct, Periods)/Periods) // Ulcer Index
//           MAE:          {%2}         Lowest(S.MAE/S.Alloc[1], Periods) // Max Adverse Excursion
//           MFE:          {%2}         Highest(S.MFE/S.Alloc[1], Periods) // Max Favorable Excursion

Exits:        {#|"Trades"} Sum(S.Exits, Periods) // use "Trades" (a reserved word) as column header
_wins:        Sum(S.Wins, Periods)
_lossses:     Sum(S.Losses, Periods)
PctWins:      {%2}        _wins/Exits

// position-size-based trade-level summary stats
AvgWin:       {%2}        Sum(S.WinPct, Periods) / _wins
AvgLoss:      {%2}        Sum(S.LossPct, Periods) / _lossses
WinLen:       {#2}        Sum(S.WinBars, Periods) / _wins
LossLen:      {#2}        Sum(S.LossBars, Periods) / _lossses
Expectancy:   {%2|}       Sum(S.TradePct, Periods) / Exits

//           TradeLen:     {#2}         Sum(S.TradeBars, Periods) / Exits

// allocation-based trade-level summary stats
//           AvgWinA:       {%3}         Sum(S.WinPctAlloc, Periods) / _wins
//           AvgLossA:      {%3}         Sum(S.LossPctAlloc, Periods) / _lossses
//           ExpectancyA:   {%3}         Sum(S.TradePctAlloc, Periods) / Exits

// dollar-based trade-level summary stats
//           AvgWinDlr:     {$2}         Sum(S.WinDlr, Periods) / _wins
//           AvgLossDlr:    {$2}         Sum(S.LossDlr, Periods) / _lossses
//           ExpectDlr:     {$-2}        Sum(S.TradeDlr, Periods) / Exits

ProfitFactor: {#2|}       Sum(S.WinDlr, Periods) / Sum(S.LossDlr, Periods)
Sharpe:       {#2}        SQR(S.BPY)*Avg((S.NetPct-S.RiskFreeRate/S.BPY), Periods)/StdDev(S.NetPct, Periods)

//           Skew:         {#5}         Skewness(S.NetPct, periods)
//           Sortino:       {#2}         SQR(S.BPY)*Avg(S.NetPct, Periods) / StdDev(Min(0, S.NetPct), Periods)
//           Skipped:       {%2}         (1-Exits/Sum(S.Setups, Periods))

AvgUse:       {%2}        Avg(S.Usage, Periods) // max intraday capital usage
MaxUse:       {%2}        HHV(S.Usage, Periods) // max overnight total exposure

// trade statistics -- may be slow to calculate for tests with large trade counts
// Dave Bergstrom's "Edge Ratio"
//           _AvgMFE:       TradeStatAvg(if(T.Side > 0, T.Highest - T.PriceIn, T.PriceIn - T.Lowest) / T.ValueIn) //
//           _AvgMAE:       TradeStatAvg(if(T.Side > 0, T.PriceIn - T.Lowest, T.Highest - T.PriceIn) / T.ValueIn) //
//           EdgeRatio:     {#}          _AvgMFE / _AvgMAE
//           Van Tharp's "System Quality Number" (aka "T Statistic")
//           TharpSQN:      {#}          SQR(TradeStatSum(1)) * TradeStatAvg(T.NetPct) / TradeStatStdDev(T.NetPct)
// equivalent to SQR(Exits) * Expectancy / TradeStatStdDev(T.NetPct)
```

This default script includes several commented-out column definitions. Feel free to uncomment them if

you'd like to add them to your stats display.

To change the order of the columns in the window, simply rearrange the lines of this script and then re-apply it (press F4).

See **Results Section** for more details on results window customization.

At any time, there can only be one set of results column definitions, which remain active until they are replaced.

Whenever a script is run, if it contains a "Results" section, this becomes the active set of columns for all open results windows.

If the script has no Results section, then the default RESULTS.RTS will be used instead. (This same mechanism applies to Graphs, Charts and Trades customization as well.)

The summary stats displayed in the results window are always the combined results for all the strategies within the script that was run.

To view the per-strategy summary statistics, select a results row and then view the **Summary Report**.

The **Results Menu**, accessible by right-clicking in a *Results Window* or via the menu bar, provides access to all the features of this window type, including some not mentioned above.

## 7.5.2. Trade List Windows

---

The next level of test results analysis is to open the trade list, sort by various columns, and review individual trades visually on bar charts.

Right-click on a result, select "Show Trade List" (or press Ctrl+T), and then double-click on any row in that list to display the trade on a chart.

Once the chart is open, use the UP and DOWN arrow keys to quickly cycle through the trades (an oddly difficult task in most other backtesting software).

As with results and graphs windows, trade lists have an underlying script (Trades.RTS) which defines any custom columns you wish to add.

Trade List Windows include 13 standard columns plus any number of user-defined columns.

If no extra columns are specified, a trade list will show only these standard columns:



Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn	DateOut	TimeOut	QtyOut	PriceOut	Reason
00001	SPY_Crossover	SPY	Long	11/12/93	open	2,151	46.47	4/20/94	open	2,151	44.41	exit rule
00002	SPY_Crossover	SPY	Long	9/15/94	open	2,052	47.17	9/30/98	open	2,052	103.50	exit rule
00003	SPY_Crossover	SPY	Long	12/9/98	open	1,888	118.69	11/5/99	open	1,888	138.63	exit rule
00004	SPY_Crossover	SPY	Long	11/12/99	open	1,899	139.25	10/31/00	open	1,899	141.02	exit rule
00005	SPY_Crossover	SPY	Long	5/16/03	open	2,852	94.89	8/19/04	open	2,852	109.81	exit rule
00006	SPY_Crossover	SPY	Long	11/8/04	open	2,728	116.98	7/20/06	open	2,728	126.12	exit rule

Please note that as with all price and volume information that you work with in RealTest, the *QtyIn* and *PriceIn* shown are as-traded (unadjusted) values. The same is true for *QtyOut* and *PriceOut*. If there was a split between the trade entry and exit date (or a dividend when total return data is used), *QtyOut* will be different from *QtyIn*, with the prices adjusted accordingly. This is consistent with what would happen in an actual brokerage account.

The default Trades.RTS that is installed with RealTest is an example of a script which adds custom columns:

```

Active Script - C:\RealTest\Trades.rts*
▽Notes: this is the default set of additional columns for Trades windows
the first 13 columns (through "Reason") are always present and not customizable

edit this file then press F4 or click Apply to change your additional trades columns
items that begin with _ are used for intermediate calculations but not displayed

you can also add a Trades section to any script
this default one is only used when there is not a custom one

additional items that you might want are commented out below

▽Trades: // item:      {format}  formula
// Filter:      Symbol=$AAPL    // uncomment and edit this line, then
// Sort:        -DateOut, Symbol // uncomment and edit this line to sp
Bars:          {#}             T.Bars
PctGain:       {%-2}          T.Profit / (T.QtyIn * T.PriceIn * T.FxIn) / PointValue
//PctGain:     {%-2}          T.Points / T.PriceIn // old formula which does not include c
Profit:        {$-2}          T.Profit
_UP:           {#}            T.Highest - T.PriceIn
_DN:           {#}            T.PriceIn - T.Lowest
_MFE:          {#}            if(T.Side = 1, _UP, _DN)
PctMFE:        {%-2}          _MFE / T.PriceIn
_MAE:          {#}            -if(T.Side = 1, _DN, _UP)
PctMAE:        {%-2}          _MAE / T.PriceIn
Fraction:      {%-2}          T.Fraction
Size:          {$0}           T.QtyIn * T.PriceIn * T.FxIn
// Comms:      {$2}           T.CommIn + T.CommOut
// Slips:       {$2}           T.SlipIn + T.SlipOut
Dividends:     {$-2}          T.Div

// Trade FX info (for use with tests that specify a different base currency)
// FxIn:        {#4}           T.FxIn
// FxOut:       {#4}           T.FxOut
// NetFX:       {$-2}          T.NetFx

// Dollar-based MFE/MAE (for futures etc.)
// DlrMFE:      {$2}           _MFE * T.QtyIn * T.PtVal
// DlrMAE:      {$2}           _MAE * T.QtyIn * T.PtVal

```

Applying changes to this script causes the new columns to be shown in all open trade windows.

Here is how the trade list looks when scrolled horizontally to show the custom columns:

Bars	PctGain	Profit	PctMFE	PctMAE	Fraction	Size	Comm	Slip	Div
109	-3.17%	(\$3,166.27)	3.96%	-6.73%	100.00%	\$99,957	\$0.00	\$0.00	\$1,264.79
1,021	131.47%	\$127,254.78	152.78%	-5.43%	100.00%	\$96,793	\$0.00	\$0.00	\$11,665.62
229	18.05%	\$40,457.95	19.85%	-4.16%	100.00%	\$224,087	\$0.00	\$0.00	\$2,811.23
244	2.31%	\$6,099.59	11.85%	-6.53%	100.00%	\$264,436	\$0.00	\$0.00	\$2,738.36
317	17.92%	\$48,498.26	23.27%	-3.48%	100.00%	\$270,626	\$0.00	\$0.00	\$5,946.42
427	11.35%	\$36,233.30	13.52%	-2.93%	100.00%	\$319,121	\$0.00	\$0.00	\$11,299.38

Note that for tests with very large trade counts (10,000+), having many custom trade column formulas will make it take a bit longer to open the trade list. In such cases, progress is displayed in the status bar, and you can hit ESC if you don't want to wait for completion.

To calculate these formula-based trade columns, the currently loaded data file must include at least one symbol with a date range that encompasses the trade entry and exit dates. If the formula references any bar data (Open, Close, etc.) then that specific symbol must be present in memory. If a formula cannot be evaluated using currently available data, "n/a" is displayed.

One special column, *Filter*, can be used as needed to prune the trade list when you only want to see some of the trades.

Here's part of a trade list from a system with 7 strategies:

Trade	Strategy	Symbol	Side	Dateln	Timeln	Qtyln	Priceln
00001	LTF2	GD	Long	1/2/15	open	18	138.47
00002	LTF2	INTU	Long	1/2/15	open	27	92.21
00003	LTF2	NTAP	Long	1/2/15	open	60	41.68
00004	SMR1	KITE_201710	Short	1/5/15	open	65	64.11
00005	LTF2	GMCR_201603	Long	1/2/15	open	18	132.81
00006	LTF2	SYU	Long	1/2/15	open	62	39.87
00007	LTF2	CHRW	Long	1/6/15	open	35	71.90
00008	LMR1	LOCK_201702	Long	1/5/15	intraday	157	14.7777
00009	SMR2	NDRM_201710	Short	1/7/15	intraday	108	16.2225
00010	SMR1	KITE_201710	Short	1/6/15	intraday	56	72.54
00011	SMR1	ARNA	Short	1/8/15	intraday	603	6.084
00012	SMR1	LOCO	Short	1/8/15	intraday	166	24.7936
00013	SMR2	FGEN	Short	1/6/15	intraday	117	35.4795
00014	LTF1	AVNR_201501	Long	1/2/15	open	147	16.96
00015	SMR2	BLCM	Short	1/7/15	intraday	103	27.4995
00016	LTF2	APTV	Long	1/6/15	open	35	69.51
00017	SMR1	NPSP_201502	Short	1/8/15	intraday	96	42.8792
00018	SMR1	SONC_201812	Short	1/8/15	intraday	133	30.9088
00019	SMR2	HALO	Short	1/8/15	intraday	317	13.146
00020	LTF2	BABA	Long	1/2/15	open	24	104.24
00021	LTF2	INTU	Long	1/7/15	open	28	87.97
00022	LTF2	NTAP	Long	1/6/15	open	61	40.05
00023	LTF2	BK	Long	1/7/15	open	64	38.69

Now we open Trades.RTS by pressing F9 and add this filter:

Active Script - C:\REALTEST\Trades.rts\*

Trades:  
 Filter: T.Strat = 2 and T.PriceIn >= 100  
 Dava: T.Dava

Then press F4 to apply, and now the top of the list looks like this:

Trade	Strategy	Symbol	Side	Dateln	Timeln	Qtyln	Priceln
00036	SMR1	PCYC_201505	Short	1/20/15	intraday	27	151.3304
00045	SMR1	NFLX	Short	1/22/15	intraday	9	425.6512
00052	SMR1	PCYC_201505	Short	1/26/15	intraday	25	164.892
00105	SMR1	PCYC_201505	Short	2/25/15	intraday	21	195.988
00127	SMR1	ICPT	Short	3/4/15	intraday	16	257.2544
00156	SMR1	BMRN	Short	3/13/15	intraday	34	120.2136
00237	SMR1	GWPH	Short	4/27/15	intraday	33	125.84
00277	SMR1	GEVA_201506	Short	5/7/15	intraday	15	211.5256
00315	SMR1	AVGO	Short	5/29/15	intraday	30	148.0752
00325	SMR1	AMBA	Short	6/8/15	intraday	41	107.4632
00351	SMR1	AMBA	Short	6/18/15	intraday	34	127.1192
00402	SMR1	RARE	Short	7/15/15	intraday	34	131.8408
00410	SMR1	ANAC_201606	Short	7/17/15	intraday	32	144.30
00420	SMR1	AMBA	Short	7/22/15	intraday	38	122.9072
00427	SMR1	ATHN_201902	Short	7/27/15	intraday	34	134.1288
00448	SMR1	SKX	Short	8/5/15	intraday	29	160.056
00530	SMR1	CLVS	Short	9/17/15	intraday	40	113.0584
00594	SMR1	AGN_202005	Short	10/30/15	intraday	14	316.5552
00598	SMR1	UTHR	Short	11/2/15	intraday	29	152.4952
00601	SMR1	LNKD_201612	Short	11/3/15	intraday	17	250.12
00616	SMR1	ARG_201605	Short	11/17/15	intraday	40	110.4272
00753	SMR1	MKTX	Short	1/29/16	intraday	38	116.4488
00894	SMR1	ICPT	Short	4/5/16	open	33	143.47

When there's a filter formula in the Trades section, it applies to every trade list window, so you'll want

to comment it out again when done looking at that view.

The other special element of the Trades section is *Sort*, as shown in this example:

Strategy	Sy...	Side	DateIn	TimeIn	QtyIn	PriceIn	Date...	TimeOut	QtyOut	PriceOut	Reason	Bars	PctGain
ndx_momentum	ALGN	Long	7/1/21	open	98	609.31	7/21/21	close	98	619.65	end of test	13	1.70%
ndx_momentum	AMAT	Long	7/1/21	open	423	141.70	7/21/21	close	423	138.00	end of test	13	-2.61%
dow_weekly_pullback	CAT	Long	7/19/21	open	296	202.27	7/21/21	close	296	210.73	end of test	2	4.18%
dow_weekly_pullback	CRM	Long	7/19/21	open	253	236.51	7/21/21	close	253	242.11	end of test	2	2.37%

```

Active Script - C:\RealTest\Trades.rts
Trades:
// Filter: Symbol=$AAPL // uncomment and edit this line, then press F4 to temporarily filter the trade list
Sort: -DateOut, Symbol // uncomment and edit this line to specify default initial sorting (use visible column names from trades window)

```

If *Sort* is specified, every new Trade List will initially be sorted by the specified columns. Beginning a column name with a dash specifies a *descending* sort, otherwise the order will be *ascending*.

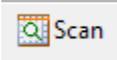
Note that the name used to refer to a column in the *Sort* statement must be its visible column title, not its formula name. Usually these are the same.

See also **Trade Comparison Windows**.

The **Trades Menu**, accessible by right-clicking in a *Trade List Window* or via the menu bar, provides access to all the features of this window type, including some not mentioned above.

## 7.5.3. Scan Windows

Scan Output Windows are very similar to **Trade List Windows** and their use is demonstrated in **Tutorial 3**.

A new scan window is created whenever a script is run in  mode, as in the sample\_scan.rts **example script**:

```

Data:
High126: Highest(C,126)
Drop: C / High126 - 1

Scan:
Filter: C > 10 and Avg(V, 20) > 100000 and Drop < -0.25
Price: {#2} C
High126: High126
Drop: {%2} Drop

```

Date	Symbol	Price	High126	Drop
6/19/20	AAL	16.00	30.47	-47.49%
6/19/20	ADS	47.40	114.03	-58.43%
6/19/20	AES	13.34	21.03	-36.57%
6/19/20	AFL	36.60	53.42	-31.49%
6/19/20	AIG	31.81	54.47	-41.60%
6/19/20	AIV	38.15	55.49	-31.25%
6/19/20	AIZ	106.28	142.52	-25.43%
6/19/20	ALK	36.29	69.34	-47.66%
6/19/20	ALLE	102.74	138.90	-26.03%
6/19/20	APA	13.98	33.59	-58.38%

Data can be sorted by any column by clicking on its header. Clicking again reverses the sort order.

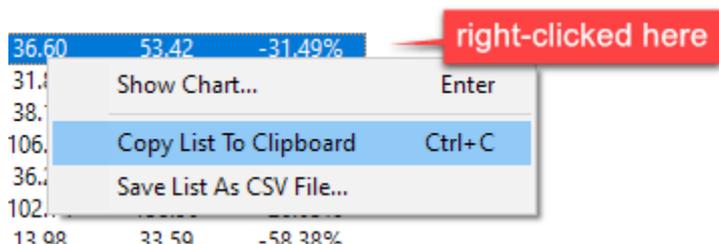
Shift-click on another column to use it as a secondary sort when values in the first sort column are the same. Shift-clicking again reverses the secondary sort direction.

Double-clicking on any row of a Scan Window opens a chart of that stock with the rightmost bar set to the date shown in that row.



Scan output can be copied to the clipboard and pasted as tab-delimited text. In Excel this automatically preserves the column structure.

Scan output can be saved to a CSV file by selecting "Save List as CSV File" from the **Scan Menu**:



The initial sort order(s) and direction(s) of Scan output can also be specified by adding a special item called *Sort* to the Scan definition, as in this example:

Active Script - C:\RealTest\Examples\industry\_indices.rts

▼ Scan:  
 Name: ?Name  
 CII: ?CII  
 List: ListNum {!-1}  
 ROC126: ROC126  
 InduRank: iif(ListNum=99,InduRank,Extern(&99, InduRank)) {!2}  
 Sort: InduRank,-List,-ROC126

Scan - 529 Items

Date	Symbol	Name	CII	▼	RO...	▼	Indu...	▲
7/21/21	SSPXREC	S&P 500 Real Estate IndGrp Index	SSPXREC	99	25.85		1	
7/21/21	EXR	Extra Space Storage Inc Common	SSPXREC	1	45.85		1	
7/21/21	IRM	Iron Mountain Inc Common	SSPXREC	1	44.85		1	
7/21/21	WELL	Welltower Inc Common	SSPXREC	1	41.62		1	
7/21/21	MAA	Mid America Apartment Communities Common	SSPXREC	1	39.50		1	
7/21/21	UDR	UDR Inc Common	SSPXREC	1	37.70		1	
7/21/21	AVB	AvalonBay Communities Common	SSPXREC	1	37.52		1	
7/21/21	EQR	Equity Residential Common	SSPXREC	1	37.36		1	
7/21/21	PSA	Public Storage Common	SSPXREC	1	37.03		1	

List one or more visible column names to specify the sort priority. Precede a name with a dash to indicate *descending* order, otherwise it will be *ascending*.

Note that the name used to refer to a column in the *Sort* statement must be its visible column title, not its formula name. Usually these are the same.

## 7.5.4. Trade Comparison Windows

Trade Comparison Windows are a special type of List Window that is created when *Compare Trade Lists* is selected from the **Results Menu**.

First a dialog box is shown to allow selection of the two Test+Strategy pairs to compare as well as other options.

Compare Trade Lists

Trade List 1  
 Test: 0001:mhp\_actual\_vs\_te  
 Strategy: live

Trade List 2  
 Test: 0001:mhp\_actual\_vs\_te  
 Strategy: test

Compare  
 Cancel

Require Exact Match  
 Entry Date  
 Exit Date  
 Strategy Name

Keep In List  
 Matches  
 Non-Matches

Minimum Absolute Difference  
 Dollars: 0  
 Percent: 0  
 Use Account Percents

Any Test+Strategy from the active **Results Window** can be selected for either trade list. You might want to compare the same strategy between two tests that had slightly different parameters, for example. In the above, the backtest of a group of strategies is being compared to their live trade results for the same period.

For each trade in the first list, the best match is found from the second list.

To be considered a match, the following must always be true:

- the symbols must be the same
- there must be some overlap in the date ranges of the two trades

If any of the *Require Exact Match* options are chosen, those criteria are added to the above list.

When more than one trade from the second list matches a trade from the first list, the best match is selected based on strategy name comparison.

The *Minimum Absolute Difference* filters can optionally be used to exclude matching trades with little or no difference in net profit, expressed in dollars and/or percentage.

*Use Account Percents* determines whether PctGain figures for each trade, as displayed and as used to compute trade differences, are calculated only for the trade itself or as a percentage of the account value at the time of the trade.

The comparison output is shown in a list window like this (image was divided for readability, and this list had many more rows):

Compare - Test 1 live vs Test 1 test - Net Difference \$3.94 / 0.01%										
Date	Symbol	Strategy1	DateIn1	QtyIn1	PriceIn1	DateOut1	QtyOut1	PriceOut1	Profit1	PctGain1
1/3/22	ASO	MOCShort	1/3/22	773	45.25	1/3/22	773	44.21	\$798.36	2.28%
1/4/22	BILL	MOCLong	1/4/22	156	224.17	1/4/22	156	218.42	(\$898.30)	-2.57%
1/7/22	BYND	MOCShort	1/7/22	479	72.94	1/7/22	479	68.52	\$2,113.61	6.05%
1/4/22	CFLT	MRLong	1/4/22	531	65.83	1/6/22	531	64.94	(\$474.51)	-1.36%

Strategy2	DateIn2	QtyIn2	PriceIn2	DateOut2	QtyOut2	PriceOut2	Profit2	PctGain2	ProfitDiff	PctGainDiff
MOCShortX	1/3/22	773	45.25	1/3/22	773	44.21	\$796.19	2.28%	\$2.17	0.01%
MOCShortX	1/7/22	479	72.94	1/7/22	479	68.52	\$2,112.39	6.05%	(\$898.30)	-2.57%
MRLongX	1/4/22	531	65.83	1/6/22	531	64.71	(\$600.03)	-1.72%	\$125.52	0.36%

When a match is found, each trade's basic details are shown in the same row, along with the net profit differences.

When a match is not found, each trade gets its own row, and its Profit and PctGain become the differences for its row.

As in any **List Window**, you can sort by any column by clicking on its header.

Double-clicking on any row brings up a chart showing both trades (if there was a match):



Once in a chart, the Up and Down arrow keys can be used to cycle through the underlying list.

The **Compare Menu**, accessible by right-clicking in a *Trade Comparison Window* or via the menu bar,

provides access to all the features of this window type, including some not mentioned above.

## 7.5.5. Other List Windows

---

General-purpose *List Windows* are created when any of the following features are used:

- *Examine Data* or *Examine Stats* on the **Debug Panel**
- *Show Data* in a **Chart Window**
- *Show Data* in a **Graph Window**

List windows show data in columnar format.

Data can be sorted by clicking on the header of any column.

List window contents can be copied to the clipboard or saved to a CSV file.

## 7.6. Graphical Windows

---

RealTest includes several Graphical Window types, all of which are used when **Analyzing Test Results**.

Each Graphical Window type is described in the following sections.

### 7.6.1. Daily Stats Graphs

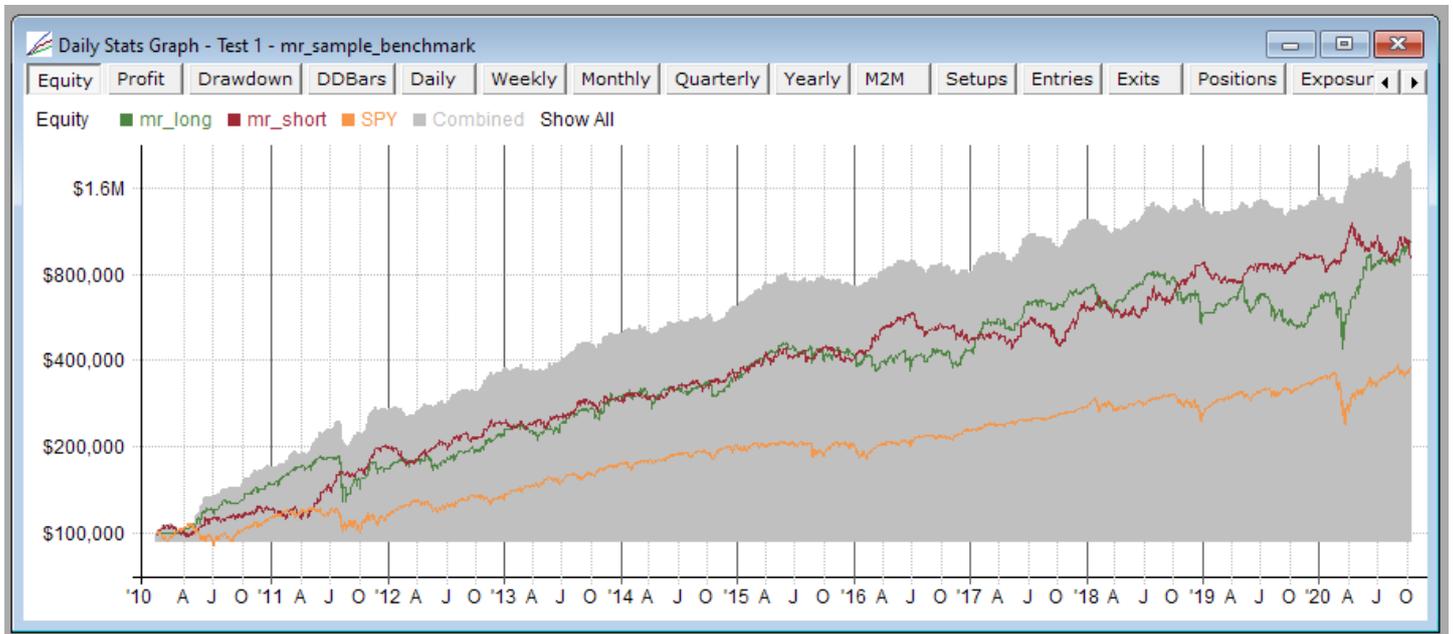
---

As we drill further down, the next level of results analysis to look at are the daily stats graphs.

Each test record in a results file includes each stat element as a daily time series rather than just a final summed value.

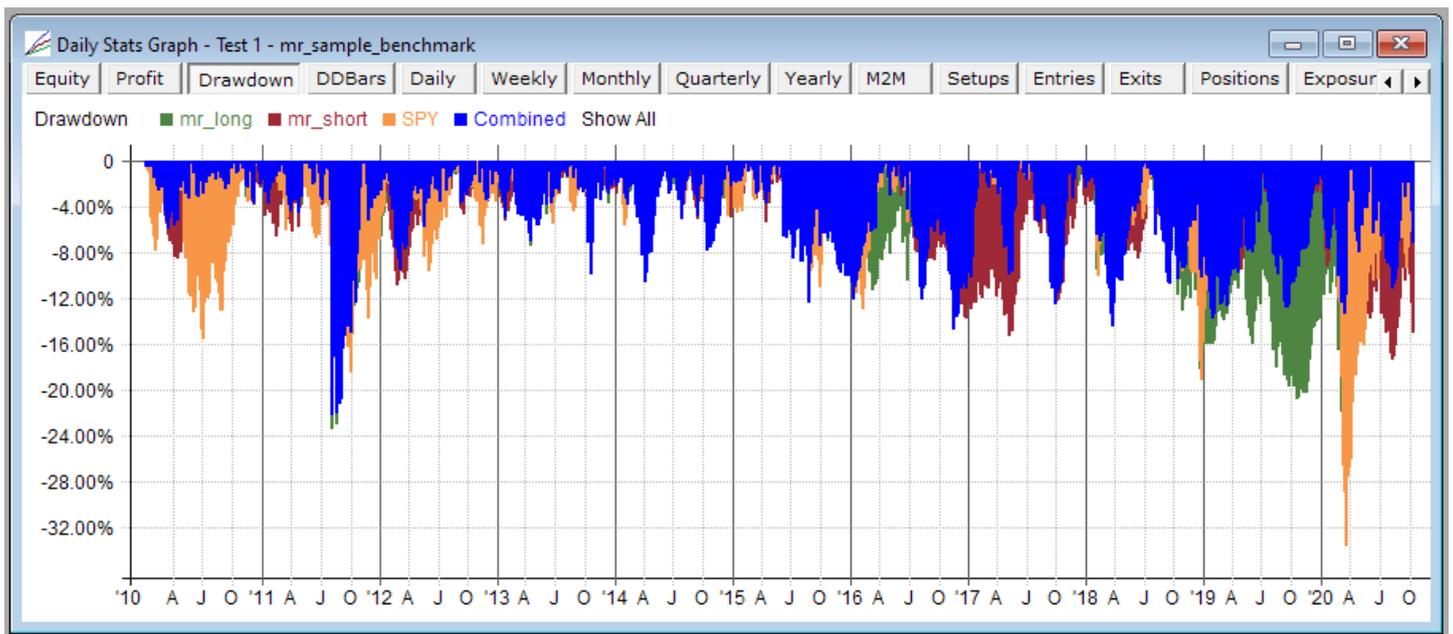
These results time series are also available to the test script while the test is running, allowing a strategy to dynamically refer to its own performance and/or the performance of other simultaneously running strategies. (The example script `mr_tracking.rts` shows how this is implemented.)

Double-clicking on a row in a results list opens the daily stats graph for that test. Here is an example from a 2-strategy system, long/short mean-reversion:



Each strategy is graphed separately with color-coded lines and labels, along with the combined system results.

Here is the drawdown graph, showing how the two sides of this long/short strategy will usually hedge each other to some degree:



Stat series lines (or bars) can be displayed for any combination of the multiple strategies that were contained in the system defined in the test script that produced this set of results.

Which strategies to display and which to hide can be specified by clicking on their names at the top of the graph:

**mr\_long**  mr\_short  SPY  Combined **Show All**

The names switch to boldface when you hover over them to show that you can click.

The square next to each name is filled if that line is showing or hollow otherwise.

Clicking on the name of a strategy hides all other strategies except for that one.

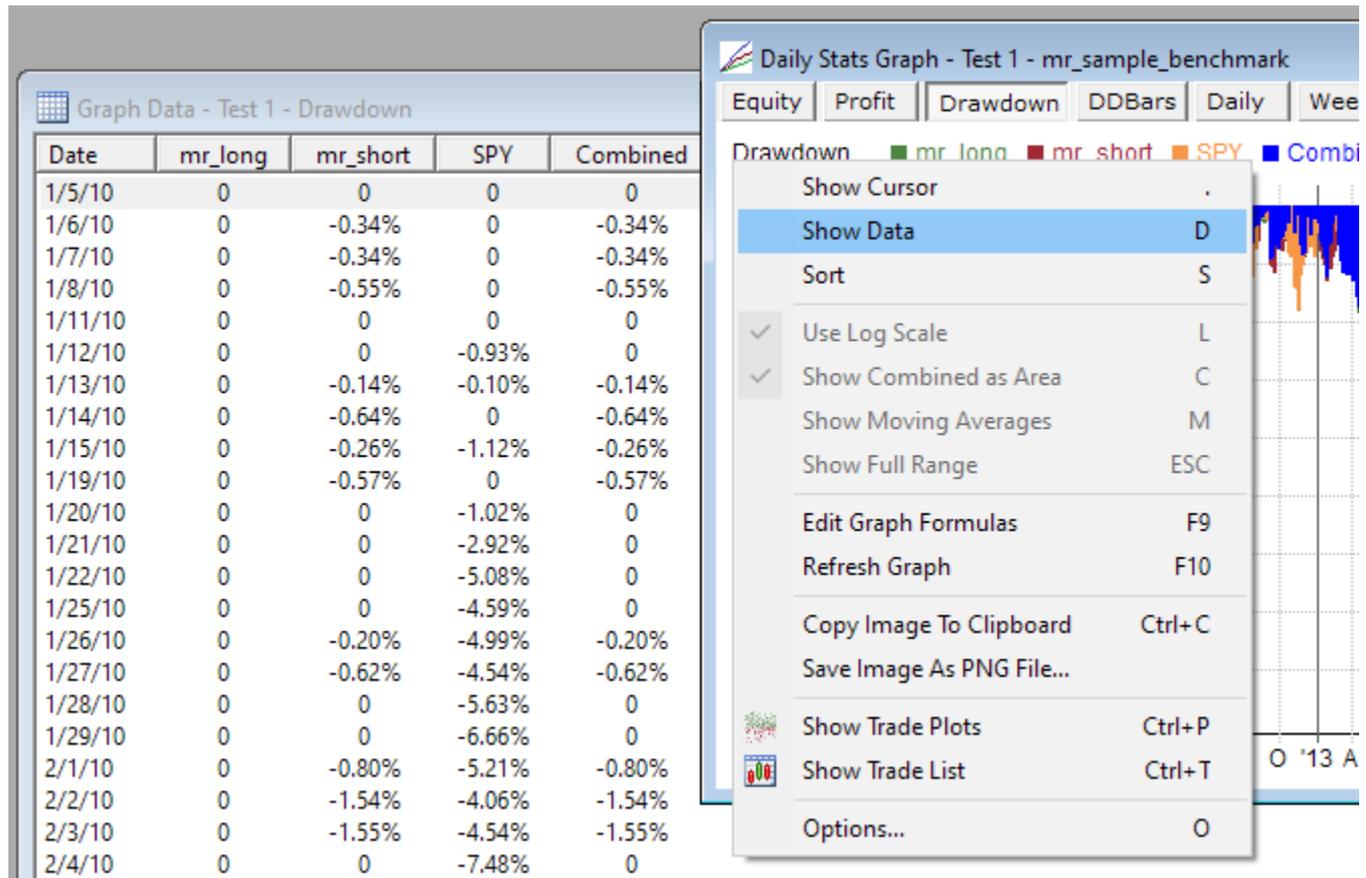
Holding down Ctrl while clicking on the name of a strategy toggles it between shown and hidden without effecting other strategies.

Clicking "Show All" shows all the strategies plus the combined line.

Holding down Ctrl while clicking "Show All" hides all the lines.

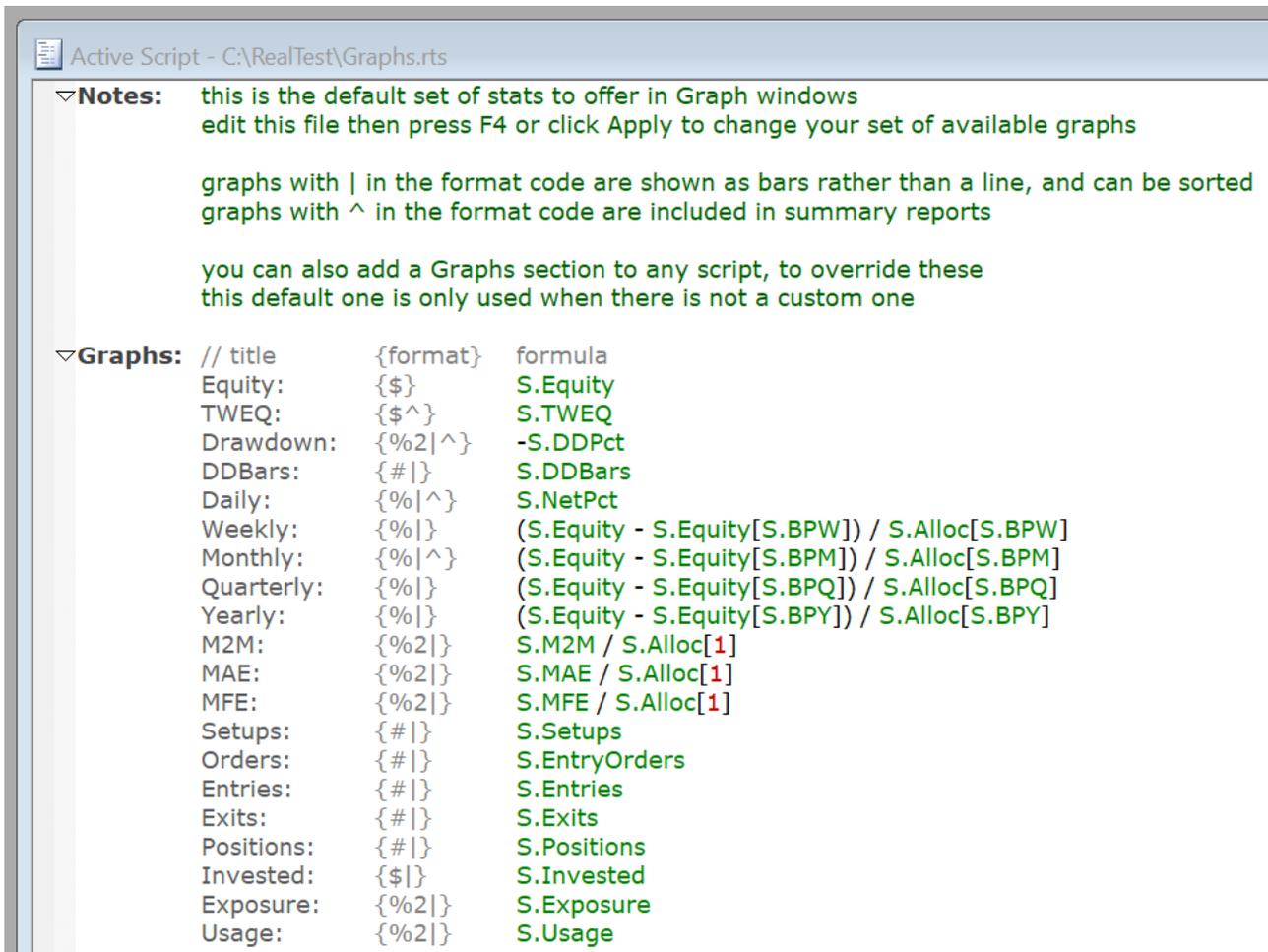
Strategies can also be selected (or toggled if Ctrl is pressed), or by using the number keys 1-9 and 0. If there are more than 10 strategies, shift+digit can be used for numbers 11-20.

The data underlying a stats graph can also be easily viewed by pressing the D key or selecting *Show Data* from the context menu:



In every stats graph, the Y axis represents the daily values resulting from evaluating the formula for that graph, and the X axis contains the dates.

As with results and charts, daily stats graph content is defined using the **Graphs Section** of a script, and all open graph windows use the same set of graph type definitions:



If the active script does not include a Graphs section, then the default script – Graphs.rts in the program directory – is applied.

To quickly access the underlying formulas for any Results, Graphs, Trades or Chart window, press the F9 key or use the context menu.

Press F4 or click Apply after modifying the graph definitions to see the result of your changes in all open graph and data table windows.

## Graph Colors

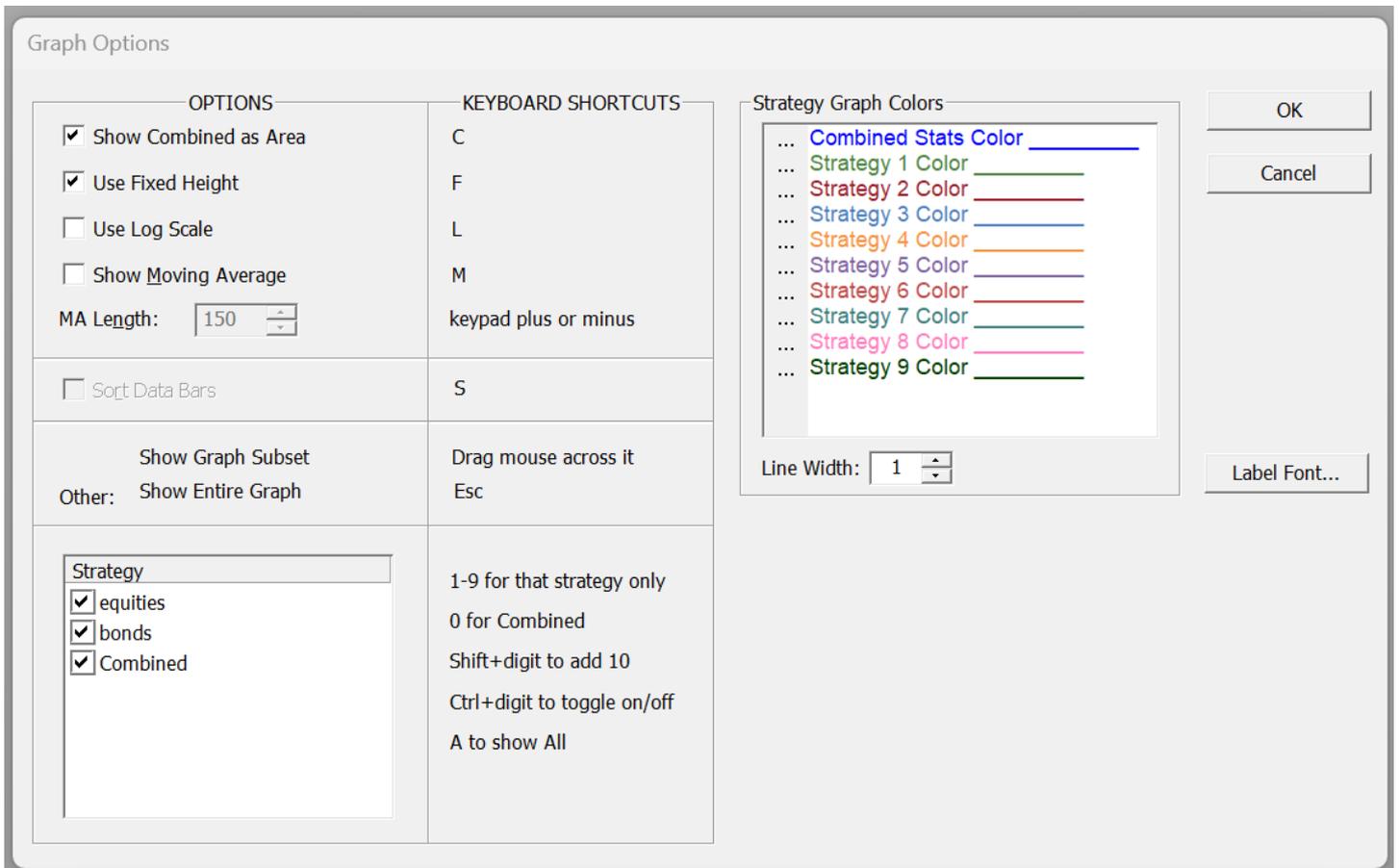
Graph windows allow specification of 10 colors via the options dialog.

The first strategy that is defined in a system script gets line color 1, the second strategy line color 2, and so on.

Lines are labeled with the name of each strategy in color-coded text, so it should always be clear which is which.

If there are more than 9 strategies, strategy 10 gets color 1, strategy 11 gets color 2, etc.

Line colors can be edited using the graph options dialog:

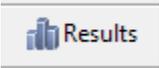


Clicking on the "." to the left of any color opens the Windows color selection dialog.

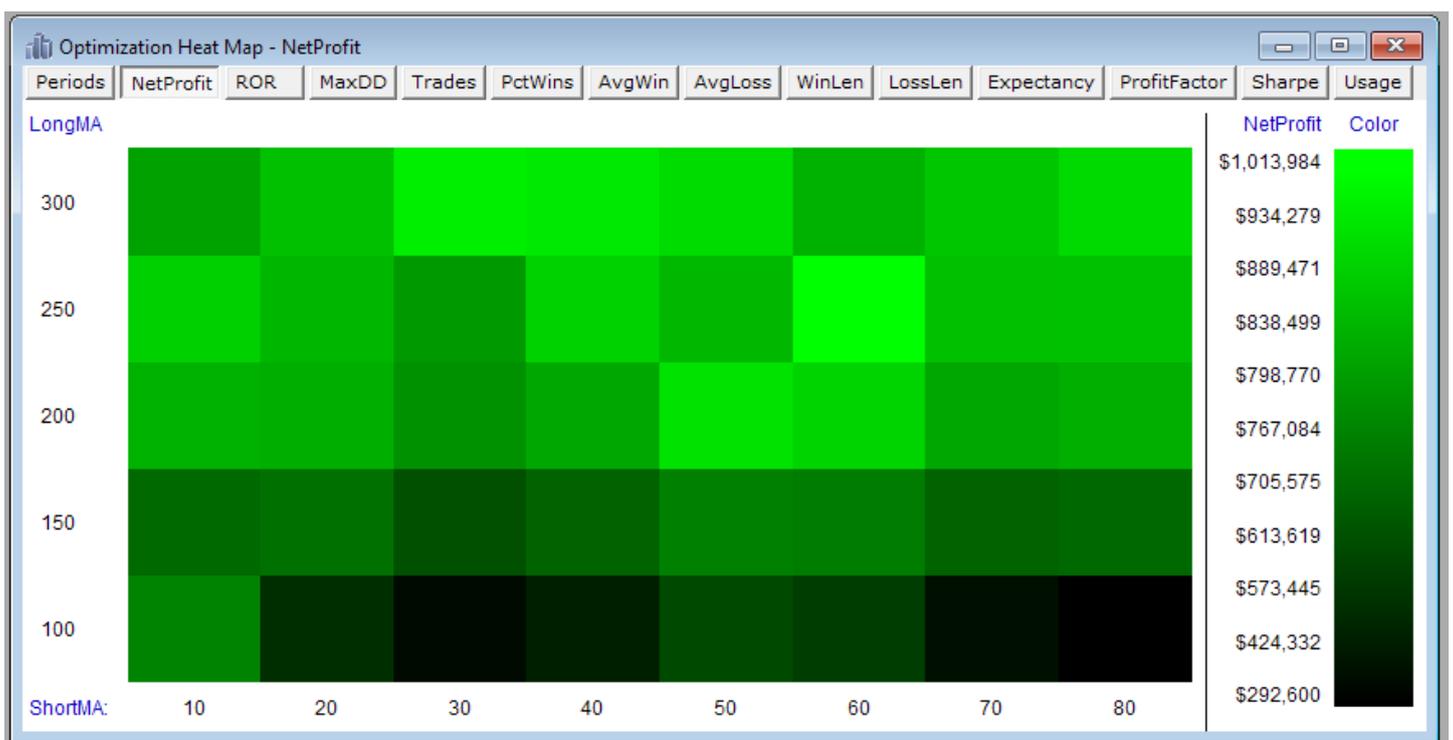
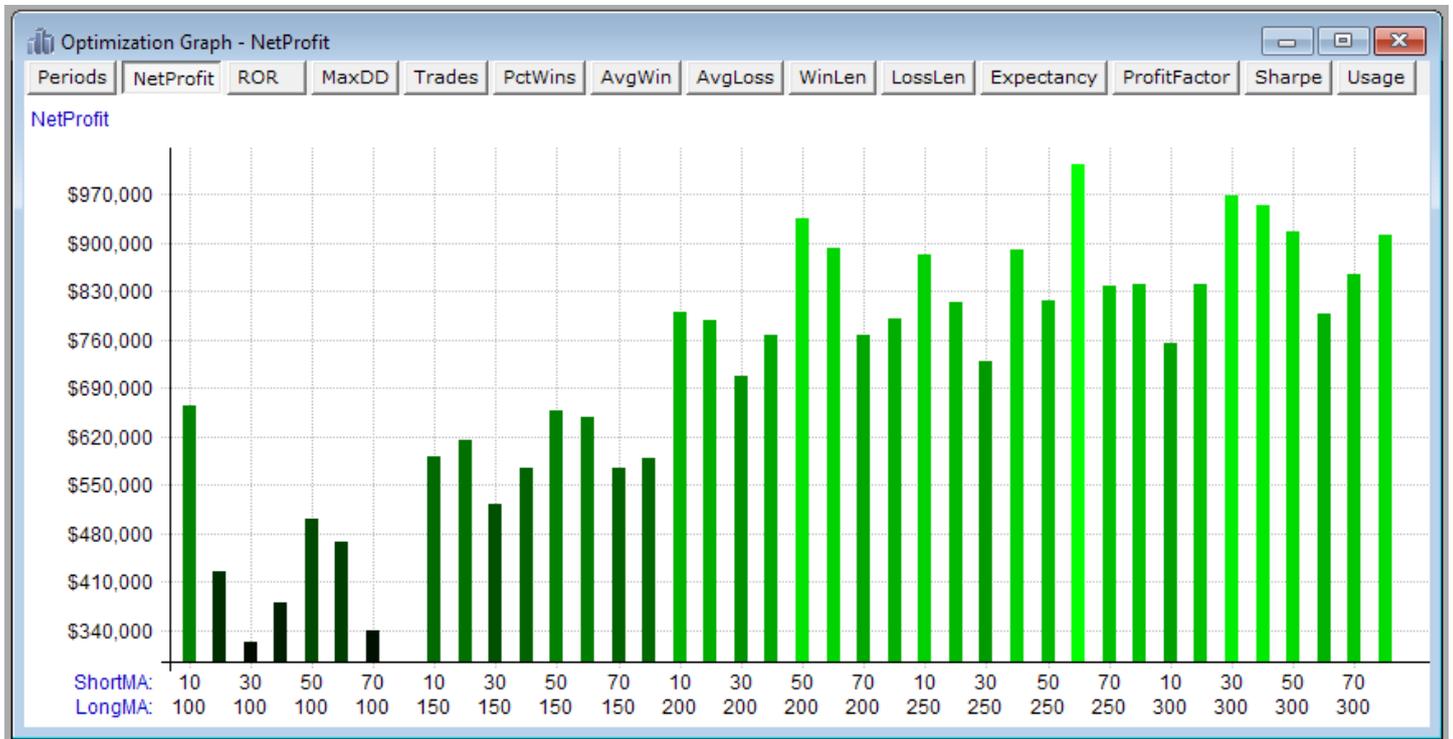
Most of the other options in this dialog can also be accessed using the **Graph Menu** and/or the keyboard shortcuts listed above and on the menu items.

This and other options dialogs in RealTest also include reminders about keyboard shortcuts that can be used.

## 7.6.2. Optimization Results Graphs

An Optimization Graph window appears when you select *Show Optimization Graph* from the **Results Menu** or press the  button on the Tool Bar.

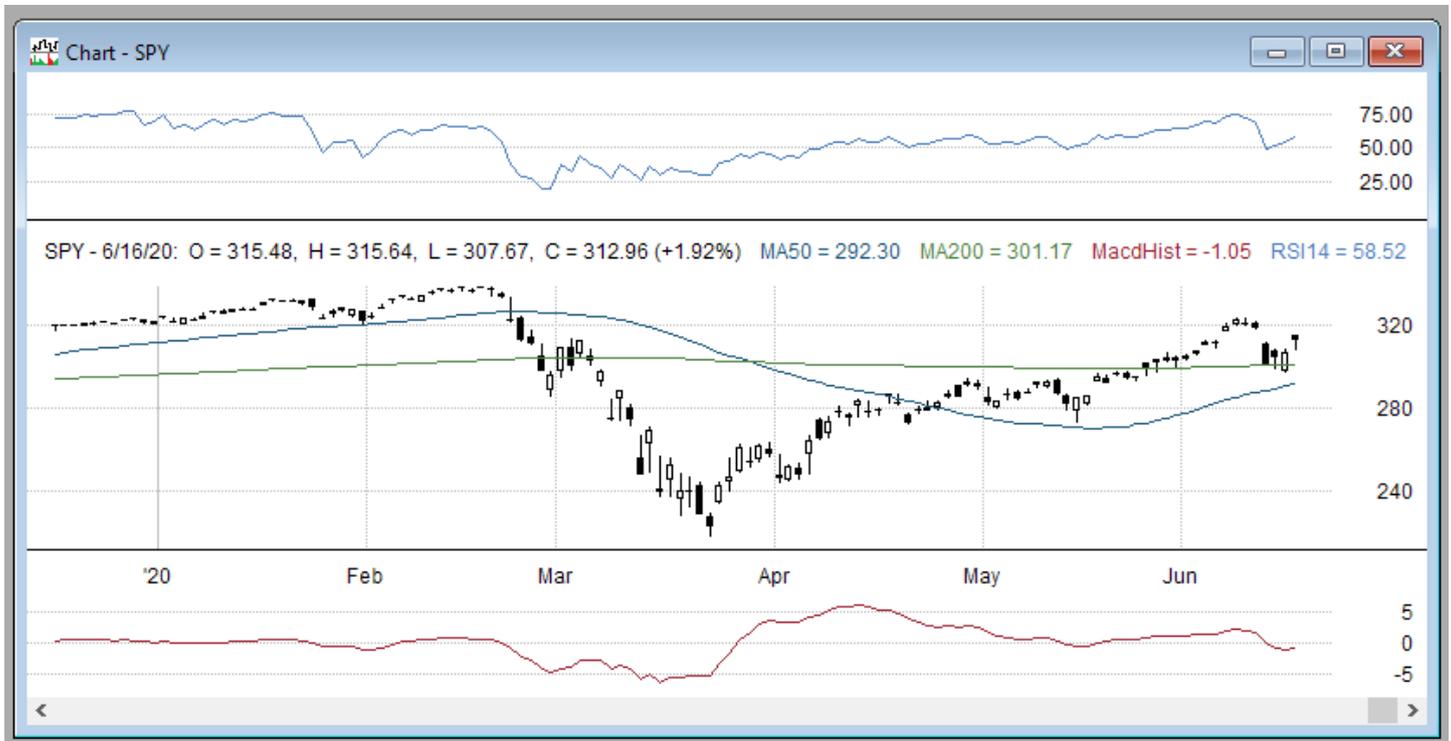
If you haven't already done so, please go through **Tutorial 2** for a detailed tour of this functionality.



### 7.6.3. Candlestick/Bar Charts

The most common way you will open chart windows is from either a trade list or scan output list.

There is also a general-purpose "Show Chart" item on the **Data Menu**, for when you just want to look at a specific symbol.



RealTest chart windows include three panes.

The top pane can show any number of indicators (but they all must share the same scale). This pane automatically appears when it has content and disappears when it does not.

The middle pane shows the OHLC bars or candlesticks, and can also show any number of price-derived lines such as moving averages or volatility bands.

The lower pane, which can also be optionally hidden, shows color-coded volume bars, or can show another group of custom indicators (sharing the same scale). This pane automatically appears when it has content or when the data includes volume bars and disappears otherwise.

The horizontal dividing line between the panes can be dragged up or down with the mouse to change the relative pane heights.

Other features of chart windows:

1. Clicking anywhere in the chart shows the cursor or moves it to that location
2. The period (dot) key toggles the cursor
3. Clicking beyond the range of bars (e.g. in the Y axis legend) or pressing ESC removes the cursor
4. The X key toggles the cross-hairs
5. Scroll Left/Right by doing any of the following:
  - left/right arrow to move 1 bar
  - shift+left/shift+right arrow to move 10 bars
  - roll the mouse wheel (with or without shift)
  - drag the Scrollbar thumb
6. Zoom in/out by doing any of the following:
  - enter a visible bar count in the options dialog
  - hold down Ctrl and roll the mouse wheel
  - press the + or - keys
  - grab and drag either end of the scrollbar thumb
  - if the caret is not visible, click and drag across a subset of the visible range to zoom in to

that subset (hit ESC to revert to the former view)

7. Drag the horizontal divider line between two panes to change their relative sizes

As with results and graphs, custom elements are added to charts using the **Charts Section** of a script:

```
Active Script - C:\RealTest\Charts.rts*
▽Notes: this is the default set of indicators for price charts (default is none)

items with ^ in the format code are show in the upper indicator pane (press 'i' to show or hide it)
items with | in the format code are shown in the lower volume pane instead of volume bars (press 'v' to show or hide it)

items that begin with _ are used for intermediate calculations but not displayed

edit this file then press F4 or click Apply to change your charts

you can also add a Charts section to any script
this default one is only used when there is not a custom one

below are examples of indicators you might want, all commented out
press Ctrl+/ within any line to toggle its comment
or click in the leftmost margin bar

▽Charts:
//      item      {format}  formula
// MA lines on chart
//      MA50:     {#}      Avg(C,50)
//      MA200:    {#}      Avg(C,200)

// MACD Histogram in volume pane
//      MacdHist: {#|}     macdh(12,26,9)

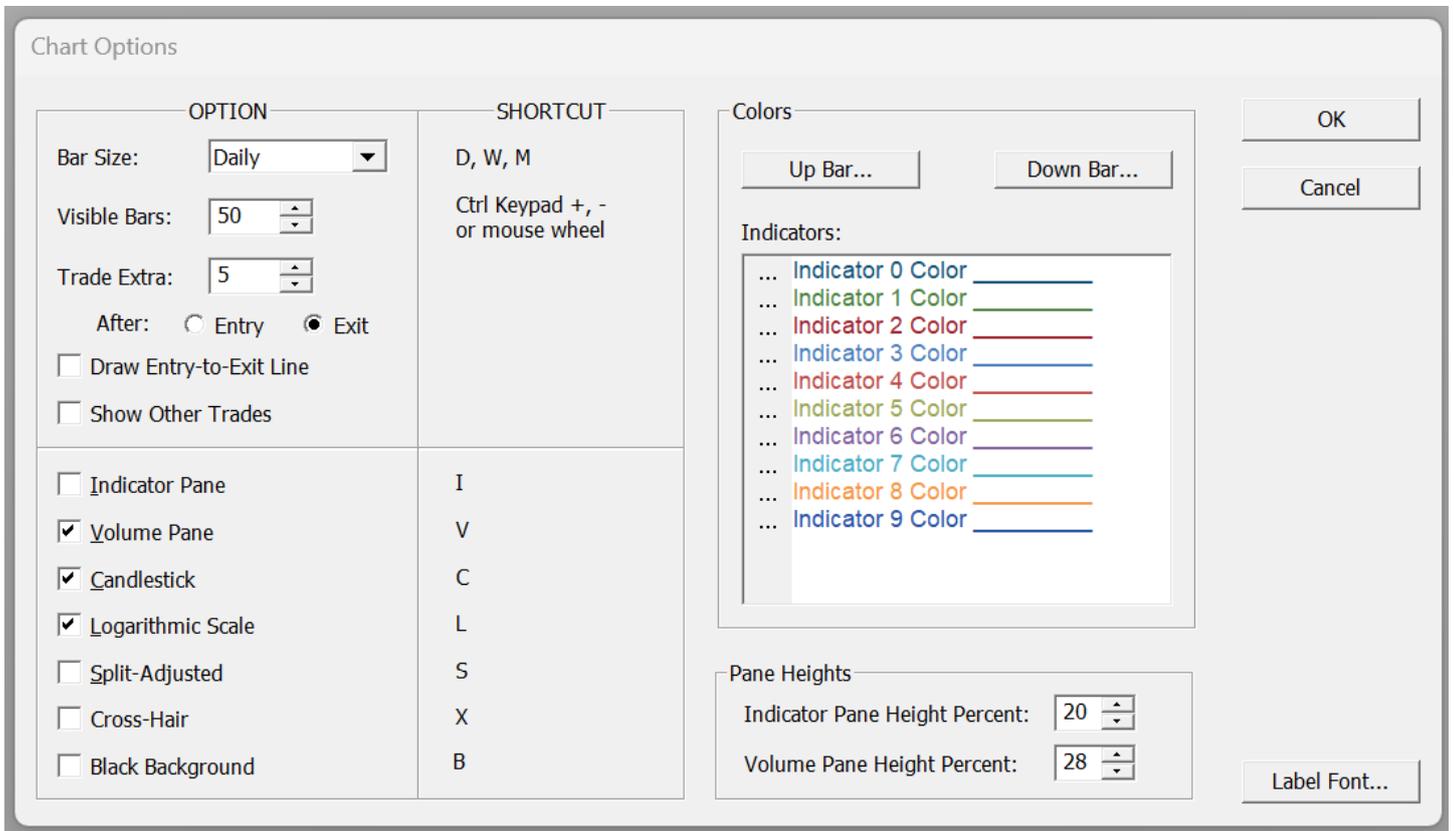
// Keltner on chart
//      _len:     20
//      _atrs:    2.5
//      _atr:     atr(_len)
//      MA20:     {#2}     avg(c,_len)
//      UB:      {#2}     MA20 + _atrs*_atr
//      LB:      {#2}     MA20 - _atrs*_atr

// RSI in indicator pane
//      RSI14:    {#2^}    rsi(14)

// Trailing Stop for Trades chart (long side example)
//      Trail:    Highest(H - 2 * ATR(14), BarsHeld)
```

If the active script does not include a Charts section then the default script – Charts.rts in the program directory – is applied.

Other chart options are accessible via the **Chart Menu** and the options dialog.



The "Trade Extra" setting controls the positioning of the X Axis relative to the current trade when scrolling through a trade list.

This makes it possible to efficiently review many trades from a backtest – an effective way to get new ideas for things to test.

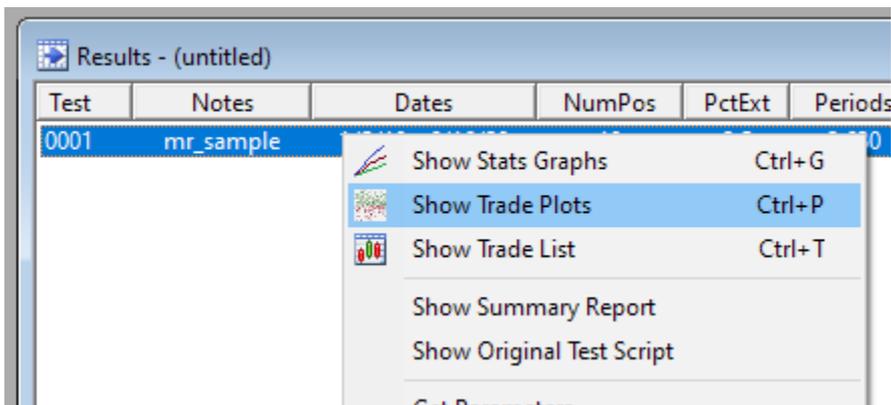
The "Show Other Trades" setting optionally causes all trades in the current symbol that are within the visible date range to be shown together on one chart.

## 7.6.4. Trade Plots and Analysis

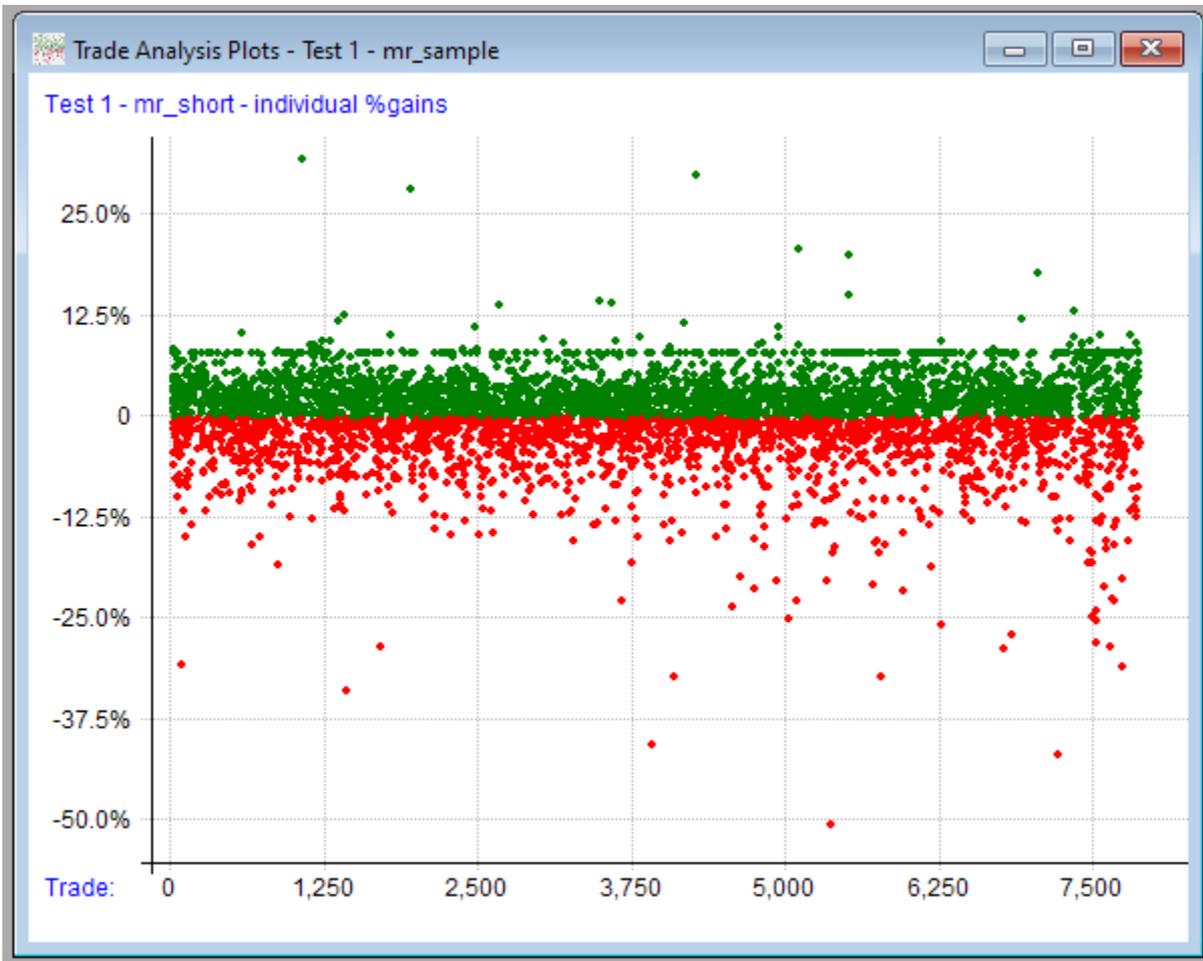
In addition to individual trade review, much can be learned by analyzing all the trades of a test (or even a huge abstract sample of artificial trades) in the aggregate.

Aggregate trade level analysis works best in systems with 1000 or more trades.

To view a trade plot and access all the trade analysis features, right-click on the test and select "Show Trade Plots".

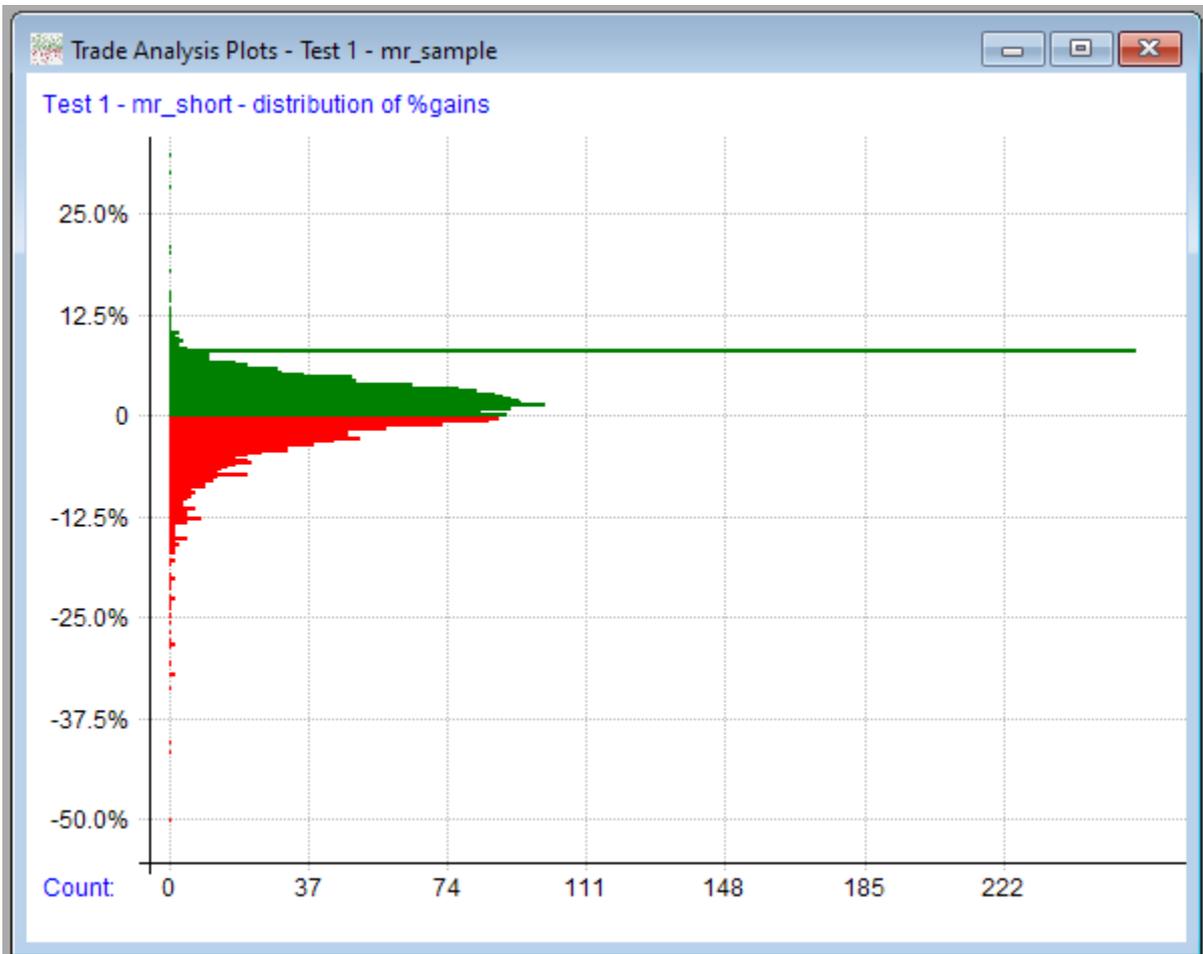


The simplest kind of trade plot is the scatter plot:

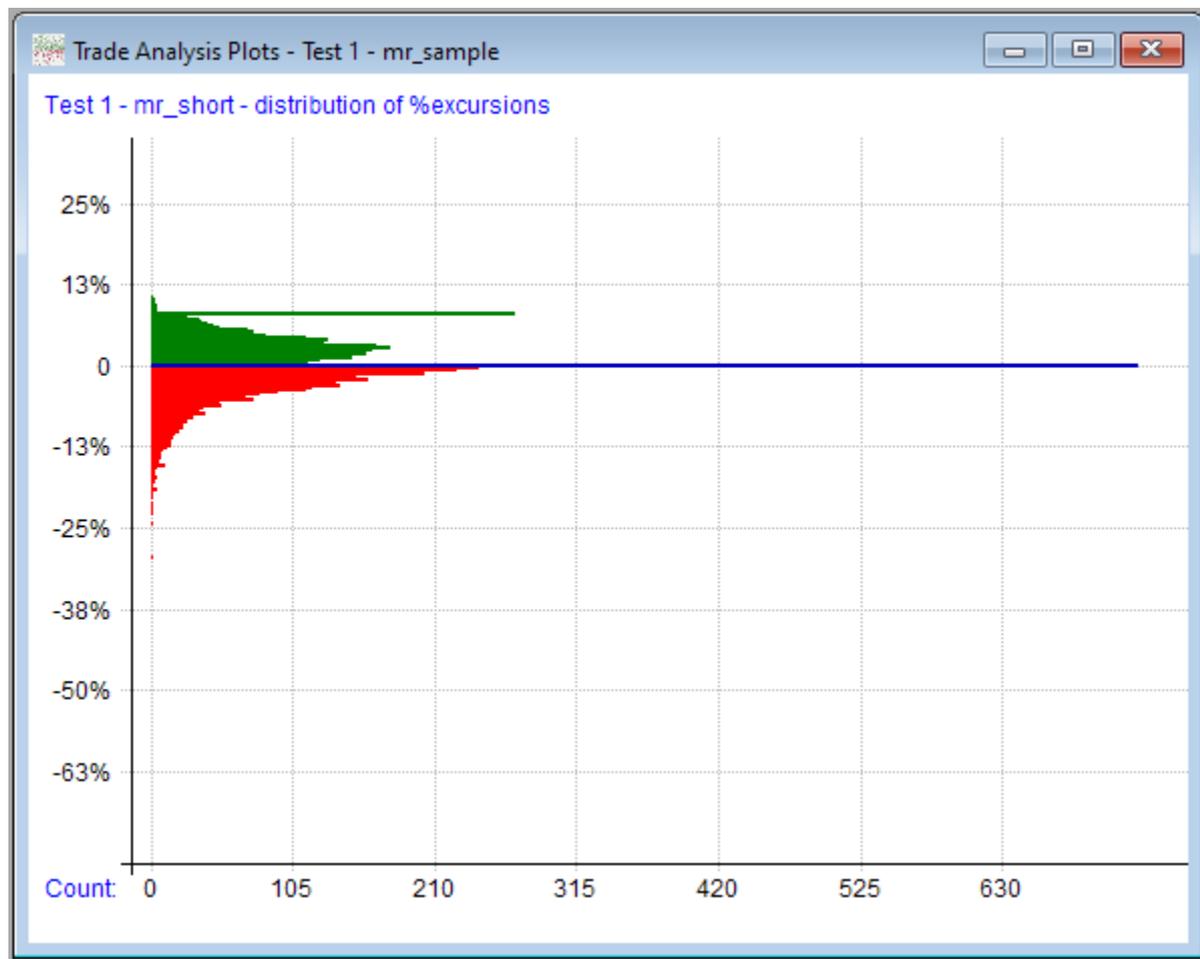


This gives a feel for distribution of trade results.

To quantify the trade distribution a bit more, we can also view it this way:



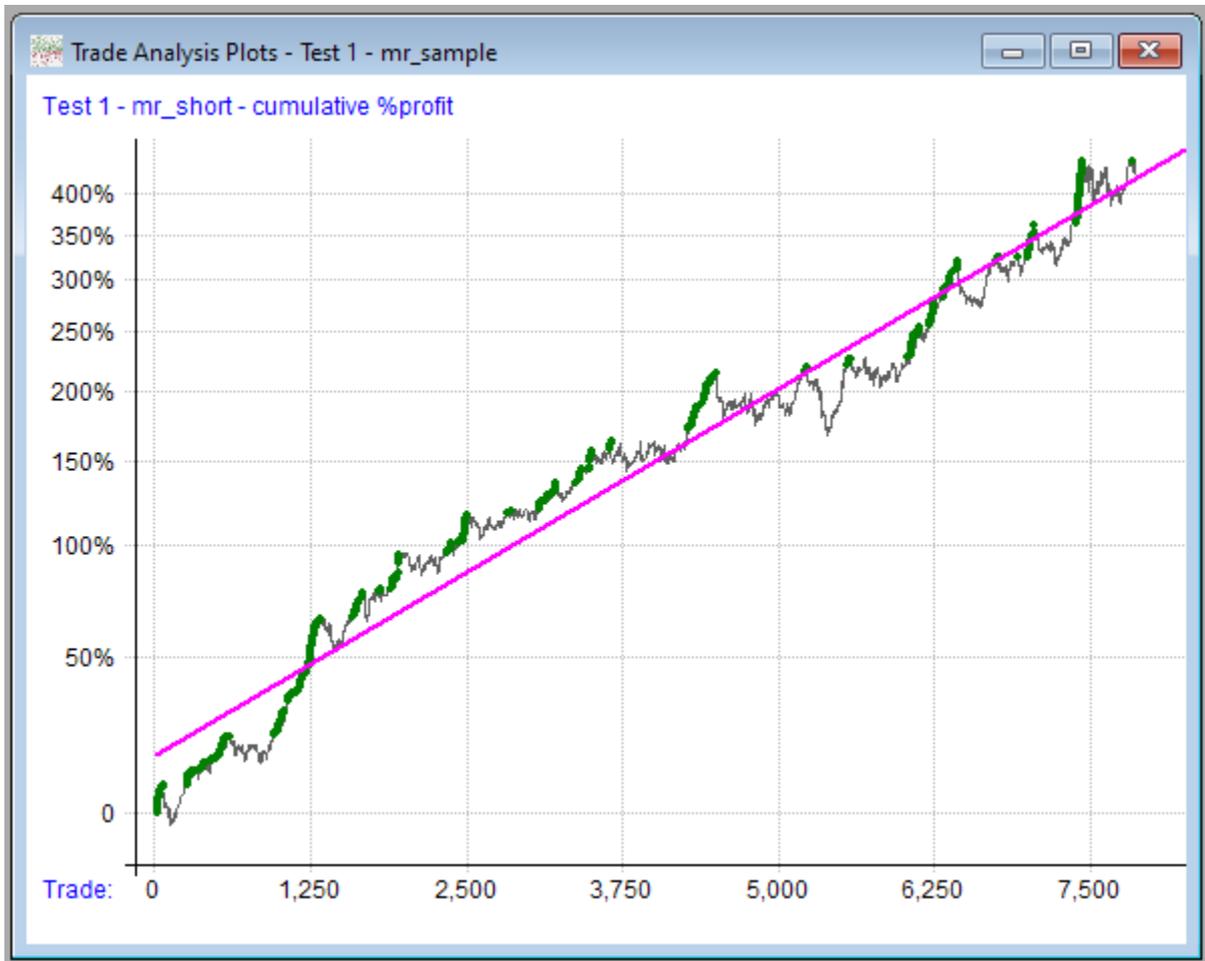
Similar to the gain distribution plot is the MAE / MFE distribution:



Values of 0 (the most of any specific value in this example) are shown in blue. (Out of about 7000 trades, approximately 10% had no favorable excursion.)

The number of distribution bins to use in the above two plot types can be specified in the plot options dialog.

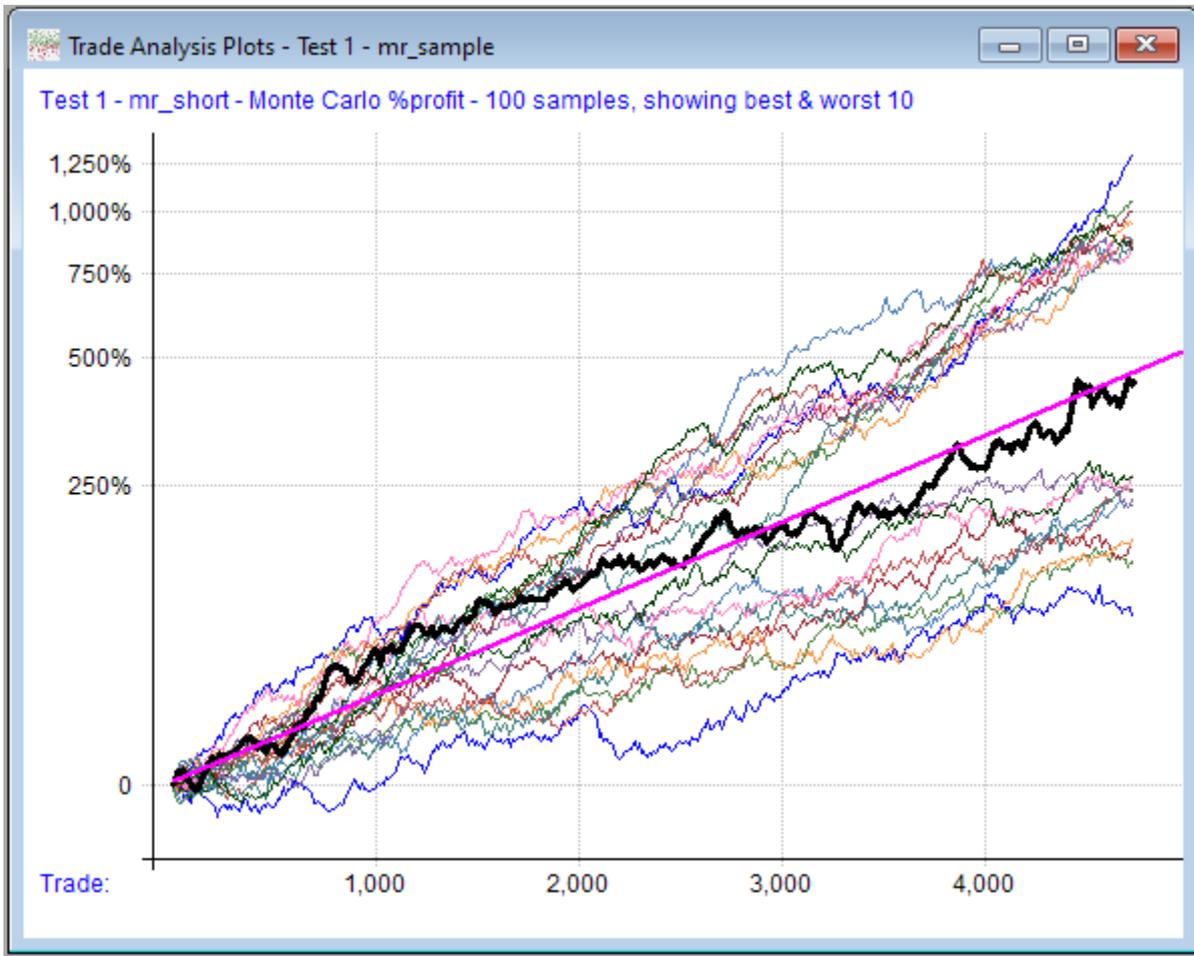
The same set of trades can also be plotted in a cumulative line with new highs marked and an optional linear regression line added:



This is the same set of dots as the first plot, with each one added to the prior one.

It will usually look something like the equity curve stats graph.

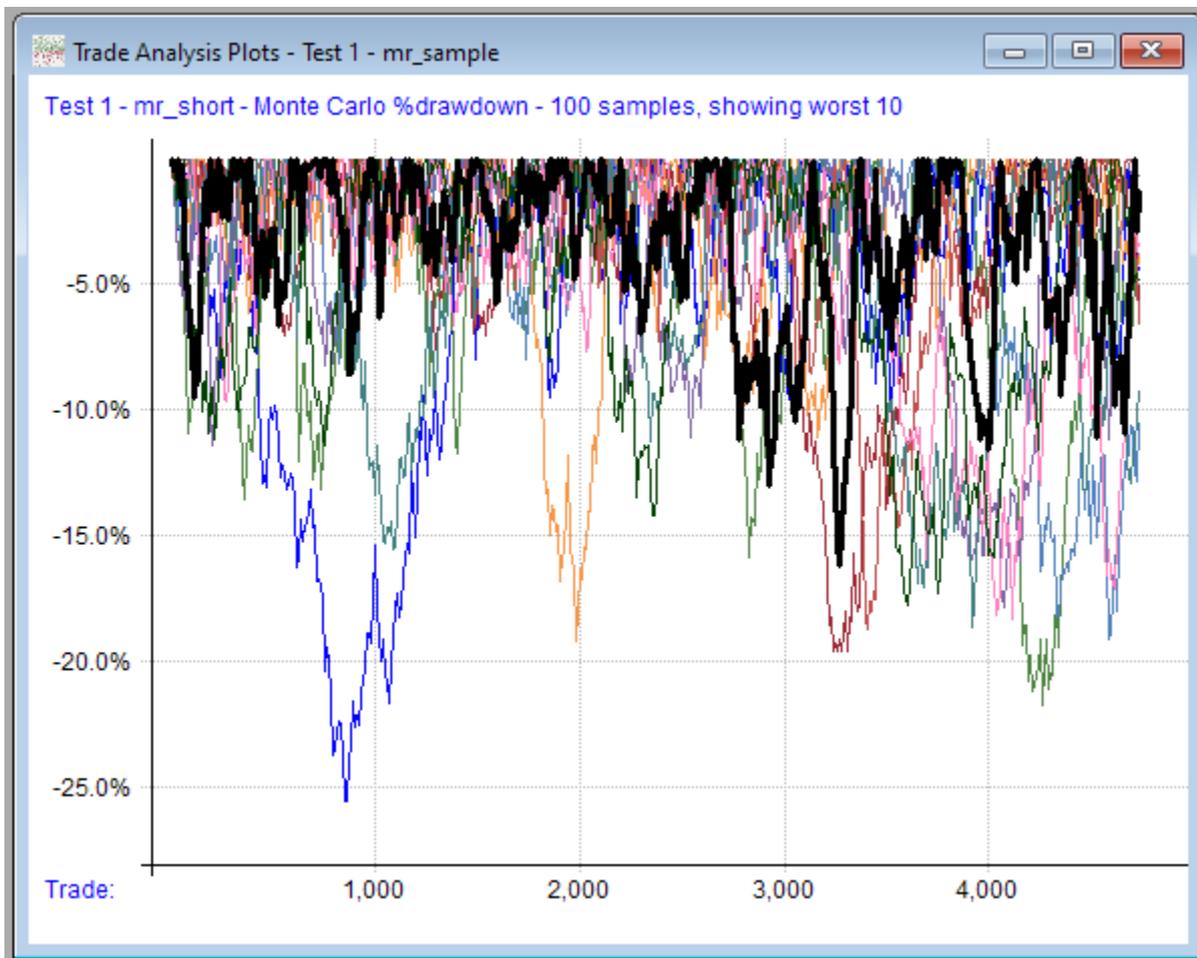
Next we have the Monte Carlo profit curves plot:



In this case, 100 random cumulative profit curves were generated by taking random samples (with replacement) from the trade list.

The original backtest profit curve is plotted in bold black over them for comparison. A linear regression line is also plotted (magenta) after being calculated from all the combined random curves.

Next we have the Monte Carlo drawdown plot:



RealTest can optionally create a table of Monte Carlo percentile stats and write it to the log window:

Log - (untitled)\*

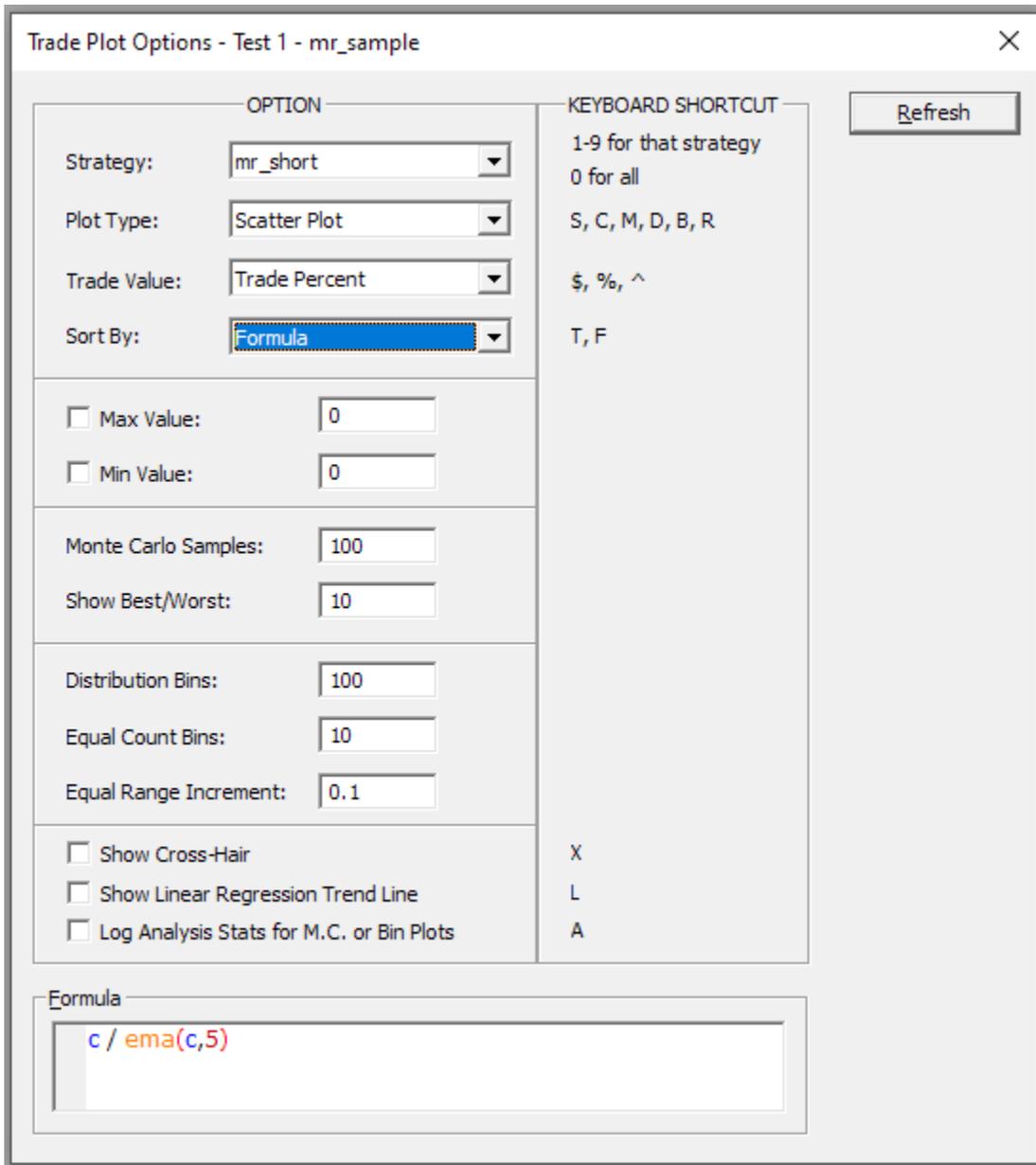
---

Monte Carlo Analysis

---

Percentile	Net Profit	CAR	Max Drawdown
1%	97.68%	6.65%	-30.24%
5%	146.05%	8.88%	-21.08%
10%	184.22%	10.37%	-18.57%
20%	218.40%	11.56%	-16.82%
50%	369.98%	15.75%	-12.31%
80%	472.56%	17.92%	-9.08%
90%	550.00%	19.35%	-7.90%
95%	672.31%	21.31%	-6.39%
99%	772.42%	22.71%	-5.35%
backtest	999.02%	25.42%	-15.99%

To understand the other two plot types, we need to first visit the plot options dialog:



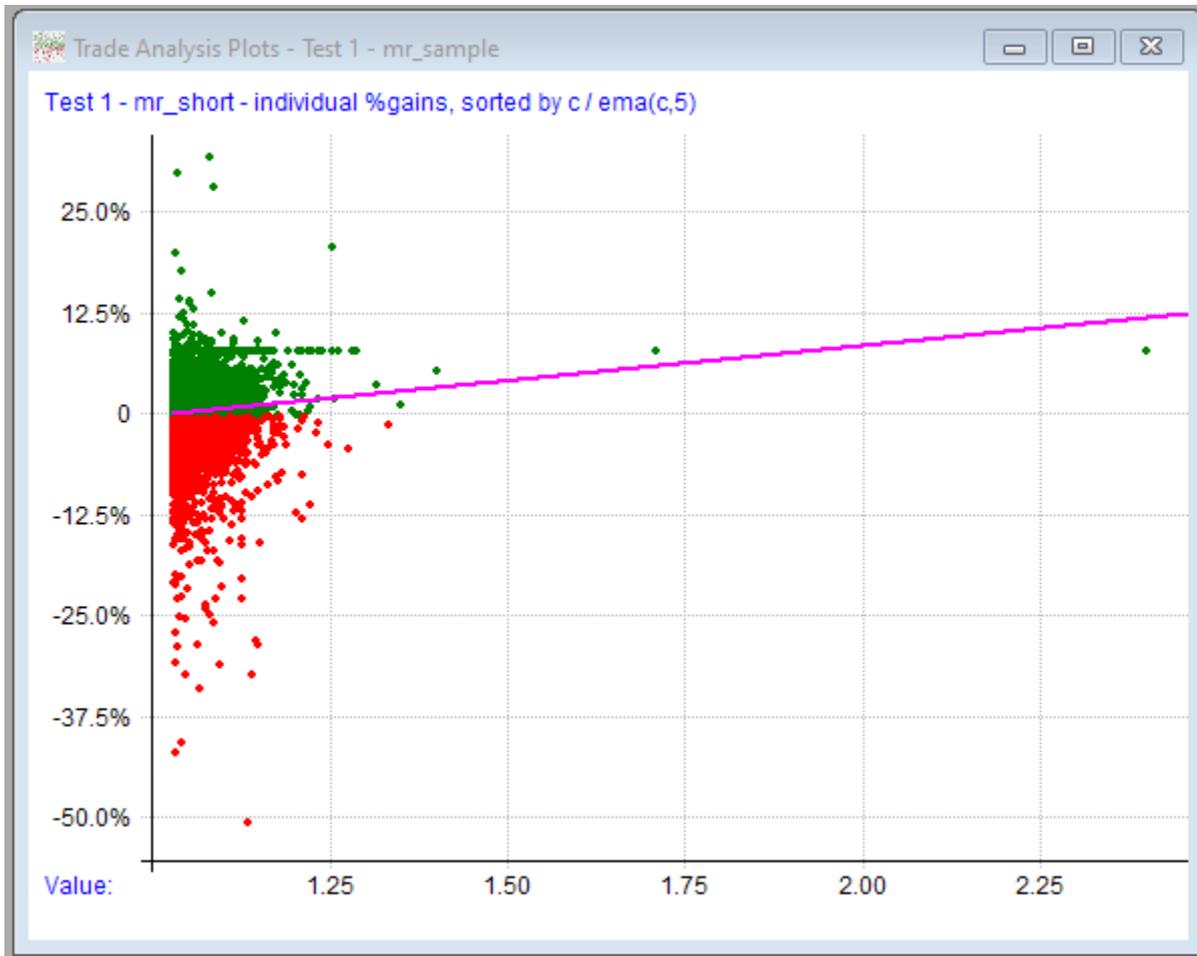
Since this dialog also serves as a research tool, it has been made "modeless", meaning you can leave it open while using other windows. (Note that if plots are opened for more than one test at once, each plot will have its own options dialog window, with the test number and name shown in the caption.)

In any of the non-cumulative plot types, the trades can optionally be sorted using any criteria you want.

The range of values from the sort formula becomes the X Axis, while the Y Axis continues to show the range of trade profits.

Below, for example, we see the scatter plot of trades sorted by distance from the 5-period exponential moving average as of the day prior to entry (in this case, the "setup" day, since entry was the next day via limit order - the formula in this dialog is evaluated using the same "current bar" as the *EntrySetup* formula for each trade.)

Clicking "Refresh" produces the following plot:



This appears to show that there's some correlation between distance above a 5-day EMA *on setup day* and positive trade outcome.

Going a level deeper, let's group the trades sorted by this formula into equal sized bins and calculate the expectancy of each bin.

Option settings:

OPTION

Strategy:

Plot Type:

Trade Value:

Sort By:

...

Distribution Bins:

Equal Count Bins:

Equal Range Increment:

Plot output:



Log output:

Log - (untitled)\*

Test 1 - mr\_short - average %gain, sorted by  $c / \text{ema}(c,5)$  - 473 trades per bin

Bin	Count	X Min	X Max	Avg Trade
1	473	1.025	1.02873	-0.00129045
2	473	1.02873	1.03254	-0.00348844
3	473	1.03254	1.03664	0.00562262
4	473	1.03664	1.04089	-0.00173412
5	473	1.04089	1.04576	0.00290953
6	473	1.04578	1.05255	0.00664183
7	473	1.05255	1.0621	0.00638616
8	473	1.06211	1.07518	0.00593839
9	473	1.07519	1.10126	0.00803085
10	473	1.10126	1.39902	0.0076753

The edge still appears to be there, especially at 4% or more above the EMA.

For a different view of the same factor, expectancy can be calculated for specific X intervals rather than equal counts:

OPTION

Strategy:

Plot Type:

Trade Value:

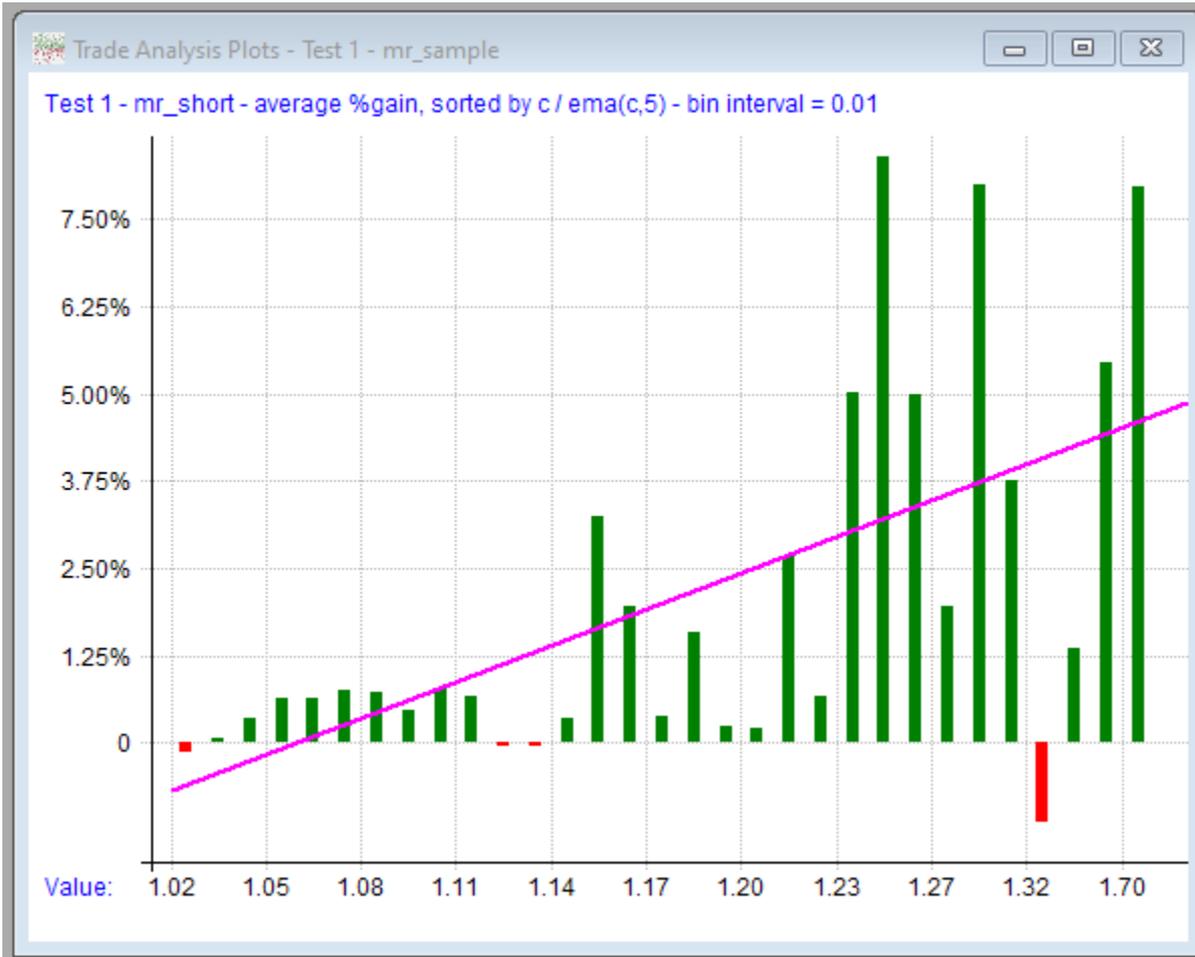
Sort By:

Distribution Bins:

Equal Count Bins:

Equal Range Increment:

Plot output:



Log output:

Log - (untitled)\*

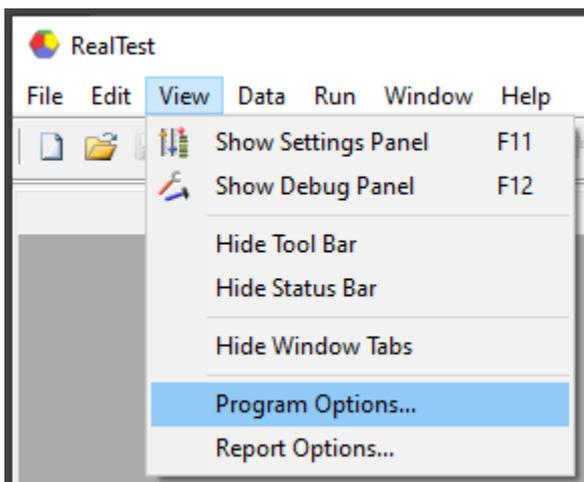
Test 1 - mr\_short - average %gain, sorted by c / ema(c,5) - bin interval = 0.01

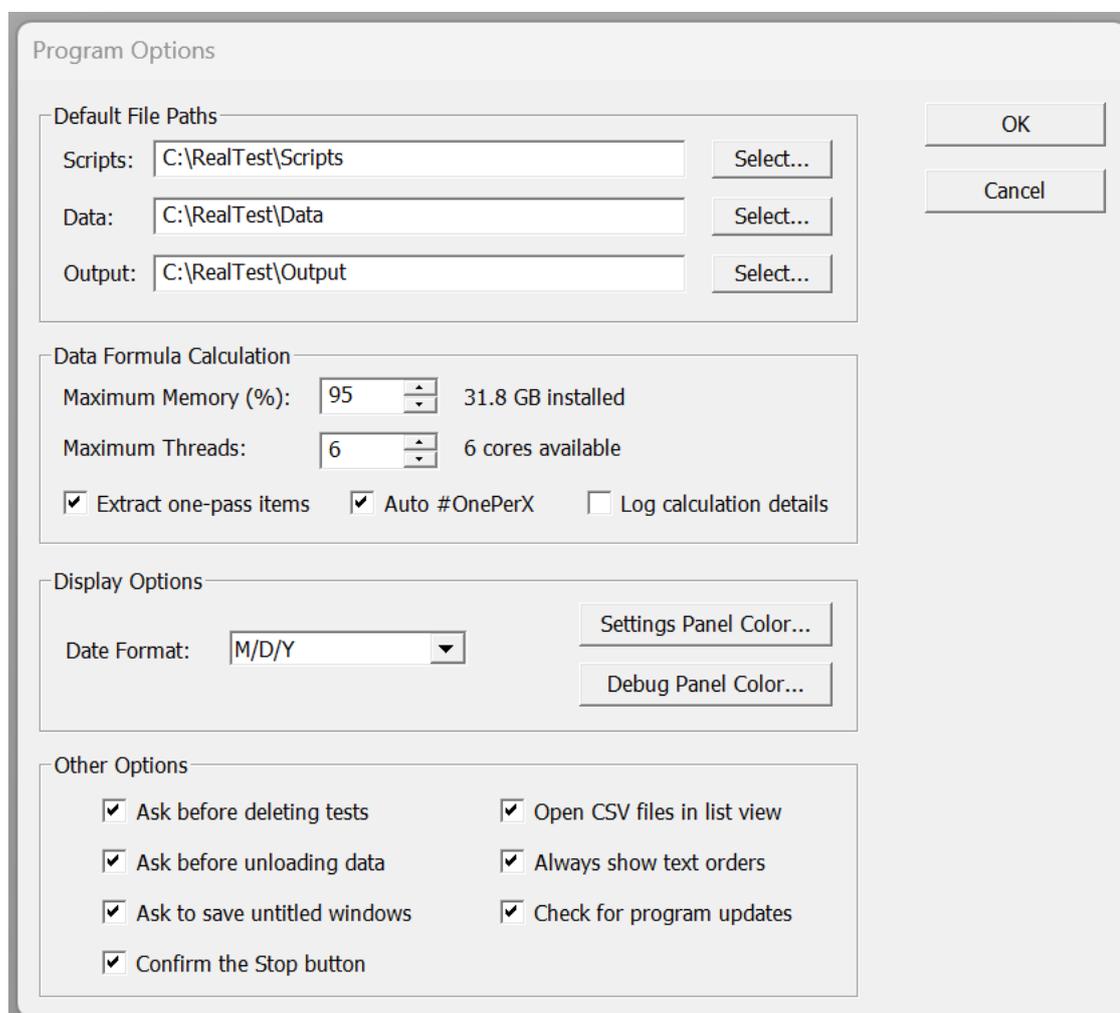
Bin	Count	X Min	X Max	Avg Trade
1	643	1.02	1.03	-0.00131546
2	1143	1.03	1.04	0.000747881
3	893	1.04	1.05	0.00352858
4	550	1.05	1.06	0.0064471
5	414	1.06	1.07	0.00648233
6	246	1.07	1.08	0.00755684
7	198	1.08	1.09	0.00725697
8	151	1.09	1.1	0.0046202
9	117	1.1	1.11	0.00787329
10	91	1.11	1.12	0.00669334
11	80	1.12	1.13	-0.000384637
12	51	1.13	1.14	-0.000605748
13	42	1.14	1.15	0.00339228
14	23	1.15	1.16	0.0323938
15	23	1.16	1.17	0.0196568
16	14	1.17	1.18	0.00367605
17	6	1.18	1.19	0.015708
18	7	1.19	1.2	0.00235618
19	12	1.2	1.21	0.00201547
20	5	1.21	1.22	0.0268356
21	4	1.22	1.23	0.00669137
22	5	1.23	1.24	0.0502685
23	3	1.24	1.25	0.0840679
24	2	1.25	1.26	0.0497978
25	2	1.27	1.28	0.0195951
26	1	1.28	1.29	0.0798319
27	1	1.31	1.32	0.037529
28	1	1.32	1.33	-0.0112187
29	1	1.34	1.35	0.013474
30	1	1.39	1.4	0.0545139
31	1	1.7	1.71	0.0796501

Note that as you move towards the right side of the plot, each bar represents fewer data points.

## 7.7. Program Options Dialog

The Program Options dialog is accessible via the **View Menu**:





Most of the options that can be specified using this dialog relate to the running of scripts:

#### ❖ **Default File Paths**

These specify the starting point to use when a script refers to a non-fully-specified (relative) file path.

The *Scripts* path applies to any non-RTD input file referenced in a script, e.g. **Include** or **TradeList**.

The *Data* path applies to any RTD file reference (whether input or output), i.e. **DataFile**.

The *Output* path applies to all non-RTD output files, e.g. **SaveScanAs**, **SaveTradesAs**, etc.

Again, all the above only apply when a partial (relative) path is used in a script. See **File Path Specification** for additional details.

Please note that whenever you decide to change a default path location, you are responsible for renaming and/or moving the actual folders yourself.

Additionally, the sub-folders *Info*, *Logs*, *Orders*, and *Reports* are automatically created under your *Output* folder, for use as the default destination of those specific output types.

#### ❖ **Maximum Memory**

Specifies a threshold above which RealTest will stop trying to allocate memory and instead will give a warning. Program stability is not guaranteed if you try to use more than 100% of installed RAM, i.e., make excessive use of virtual memory page swapping.

#### ❖ **Maximum Threads**

Specifies how many threads to create and use when running multi-threaded calculations, such as for the **Data Section**. RealTest allows up to 32 threads to be used.

#### ❖ **Extract one-pass items**

Provides a way to turn off this **Data Section** calculation speed optimization that requires more memory when used. Leave checked unless running your script causes low-memory warnings.

❖ **Auto #OnePerX**

Provides a way to disable Data Section space-saving optimization in case it is incorrect in an unforeseen item formula variation. See **#OnePerDate** and **#OnePerSym**.

❖ **Log calculation details**

If selected, the timing and other details about the calculation of each Data Section item is logged whenever calculation occurs. This can be useful to discover which items are causing calculation delays, and to see the value type and #OnePerX status of each item.

❖ **Date Format**

Specifies how dates are displayed in the user interface, trade lists, etc. Dates in imported data such as **CSV bar data** or an **Imported Trade List** can be in any recognizable format. If CSV file dates are the opposite of your display date format, use **CSVDateFmt** or **TLDateFmt** to resolve.

❖ **Panel Colors**

Lets you change the background color of the **Settings Panel** and/or **Debug Panel**.

❖ **Ask before deleting tests**

Specifies whether you want to be asked for confirmation when you press the Delete key in the Results window with a test row selected, or when you select "Delete Test" or "Delete All Tests" from the **Results Menu**.

❖ **Ask before unloading data**

Specifies whether you want to be asked for confirmation when you do anything that will cause the current data file to be unloaded from memory, such as running an **Import**, or loading a different data file (whether manually or by running a script that uses a different one). Note that you are only asked for confirmation when unloading the data would require closing one or more open windows that are using that data, such as charts or trade lists.

❖ **Ask to save untitled windows**

Specifies whether when you want to be asked to save the contents of "(untitled)" **Log**, **Script**, or **Results** windows when you try to close them.

❖ **Open CSV files in list view**

CSV files can be opened by realtest either as regular text files in a Log window, or as a structured grid in a List window. The list view makes it easier to read the contents of the file, but does not permit editing. Uncheck this box when you want to use RealTest to edit a CSV file.

❖ **Always show text orders**

When a script run in **Orders** mode, RealTest generates and displays the list of orders for tomorrow in human-readable text format. If a specific **OrdersMode** other than *Text* was specified, a separate order list file in that mode will also be generated and displayed. If you prefer to not see the text order list when also generating e.g. a CSV order file, uncheck this box.

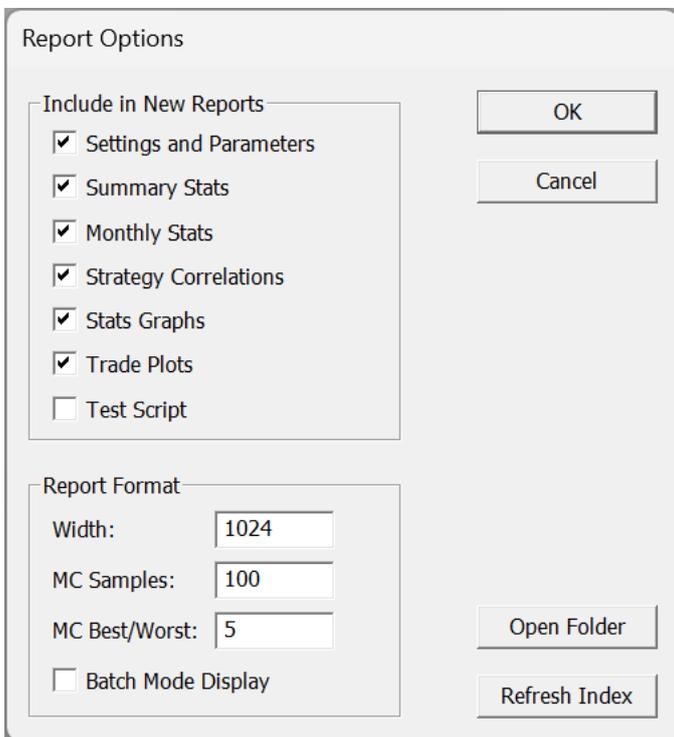
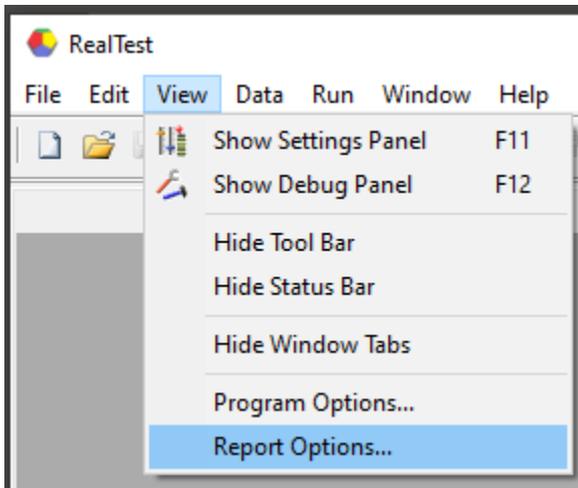
❖ **Check for updates**

Each time RealTest starts, it compares its version number to a server-based newest release number and gives you the option to download the newer release if found. Uncheck this box to disable checking for updates.

## 7.8. Report Options Dialog

---

The Report Options dialog is accessible via the **View Menu**:



All the options specified via this dialog relate to the **Test Summary Report**.

Because summary reports are static web pages (HTML and PNG files), these options only apply to newly created reports, not previously created ones:

❖ **Include in New Reports**

The items in this group let you decide whether to include or omit each available section of the report.

❖ **Report Format - Width**

Governs the maximum width constant that is placed in the generated in the HTML code.

❖ **Report Format - MC Samples and MC Best/Worst**

Specifies these options for the Monte Carlo analysis plots if those are included.

❖ **Report Format - Batch Mode Display**

Controls whether reports are opened in the browser after tests are run from the command line (batch mode).

❖ **Open Folder**

Opens the *Reports* folder, in case you want to access specific reports directly or do some cleanup.

## ❖ Refresh Index

Re-generates the *index.html* file in the *Reports* folder, which is linked to by the "Report List" link in each report.

## 7.9. RealTest Function Key Reference

---

The following are all the general-purpose function or "hot" keys available in most window types.

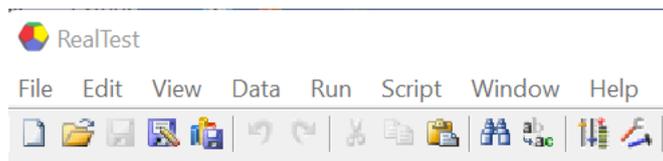
A few window types, such as **charts** and **graphs**, also have type-specific key functions, which can be found by looking at the menu and/or options dialog for those windows.

KEY	FUNCTION
Ctrl+F	Edit Find
Ctrl+H	Edit Replace
Ctrl+N	File New
Ctrl+O	File Open
Ctrl+S	File Save
Ctrl+L	File Save All
Ctrl+G	Results Show Graph (also double-click on results line or press Enter)
Ctrl+P	Results Show Plot (also works in graph windows)
Ctrl+T	Results Show Trades (also works in graph windows)
Ctrl+/ Ctrl+Break	Script Comment Toggle (also can click in left margin of line) Stops the execution of the currently running script
F1	Open the help file and show the page most relevant to what's under the cursor
F2	Show script autocomplete list, or edit test notes in results window
F3	Edit Find Next
F4	Script Apply (checks syntax, applies results/graphs/charts/trades sections, if any, to open windows)
F5	Run the active script in single-test mode (if script has strategies)
Ctrl+F5	Run the active script in order-generation mode (if script has strategies)
F6	Run the active script in multiple-test (optimization) mode (ditto)
F7	Run the Import section of the active script (if present)
F8	Run the Scan section of the active script (if present)
F9	Edit Formulas (for results/graphs/charts/trades windows)
F10	Refresh calculations (ditto)
F11	Show or hide the settings control panel
F12	Show or hide the debug control panel
Up/Down	In windows tied to lists (graph/plot->results, chart->trades/scan) changes window contents to prev/next list item (in generic chart, prev/next symbol in data file)
Left/Right	In windows with multiple views (graph/plot) changes to prev/next view, in chart shifts bars left/right or moves cursor if visible (Shift+Left/Right shifts by 10 bars)
O (letter)	In windows with an options dialog (graph/plot/chart/optgraph), opens that dialog
S	Toggles sorting in sortable graphs
ESC	Cancel calculation of trade/plot formulas, exit zoom or hide cursor in graph/chart
Home/End	Go to start or end of bars in chart
PageUp/Down	Shift chart by number of bars currently displayed (move one "page")

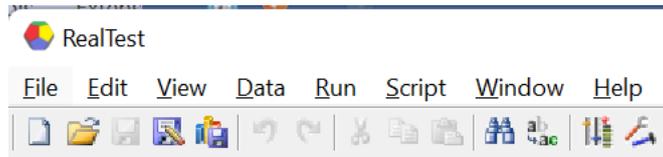
In addition to the above hotkeys, notice that most choices within the main menu and sub-menus have a letter assigned to them.

Pressing the Alt key on your keyboard reveals an underscore under each letter than can be used to select that choice.

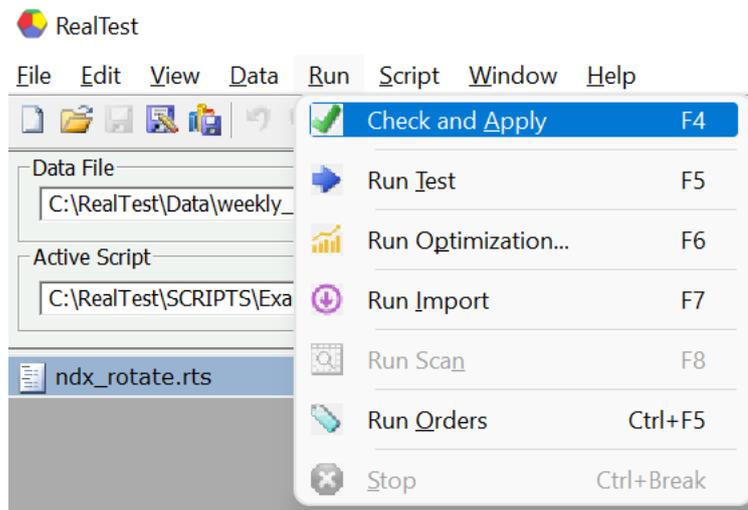
Main menu:



Main menu after pressing Alt:



Menu after pressing Alt and then R:



In this case every item on the Run menu also has a hot key assigned to it, but for example Run Test could also be initiated by pressing Alt R T.

A few others that may be useful:

File / Save As	Alt F A
View / Program Options	Alt V O
Data / Show Chart	Alt D C
Script Options	Alt S O
Window / Close All [of current type]	Alt W C

## 8. Importing Bar Data

---

RealTest uses its own memory-based binary format for daily price and volume data bars.

To get started, you must first import some data.

The data import process consists of:

- reading data from one or more external sources
- converting this external data to the RealTest binary format
- incorporating split and dividend info if provided by the external source
- optionally filtering out symbols that pass an **ExcludeIf** condition
- optionally applying additional **per-symbol information** such as company name, industry, etc.
- optionally applying **per-bar event information** such as earnings dates and amounts if available
- automatically creating weekly and monthly bars from the daily bars
- saving the imported and processed data to a local RealTestData (.RTD) disk file while also keeping it in memory for immediate use

Once a data import has been run, the saved .RTD file can simply be loaded back into memory when needed. This is much faster than having to repeatedly access an external database.

There is no need to re-import the same data. Only run an import again when you want to change the contents of the file, such as to add newer data.

When you run a test or scan, one of the **initial settings** is always a path to a .RTD file.

It often makes sense to maintain different data files for different purposes. For example, you might have several for backtesting that include delisted symbols and go back different amounts of time (5, 10, 20 years), and one where you run a fresh import every day using only currently listed stocks for running the scans that produce candidate lists for live trading.

You may also have specialized RTD files such as one with only historical Russell 3000 components, one for Nasdaq 100, etc. By selecting the data file that's most appropriate for the research you're about to do, tests can be run with maximum efficiency.

To be sure this is clear: there is no central database in RealTest, as there is in some other backtesting software. If you want to make RealTest almost as slow as other software, you could import, for example, the entire Norgate database into one large RTD file and always use that for all your work. However, I think you'll find it much more pleasant to work with multiple specialized .RTD files.

To run an import, you must open or create a script with an **Import Section**, and then run it in Import Mode.

To manually select and load an existing data file or get information about the currently loaded data file, use the **Data Menu**.

### 8.1. Norgate Data Import

---

RealTest is fully integrated with **Norgate Data** as a "3rd-Party [that's me] Supported Plugin".

Norgate is the preferred data provider for use with this software.

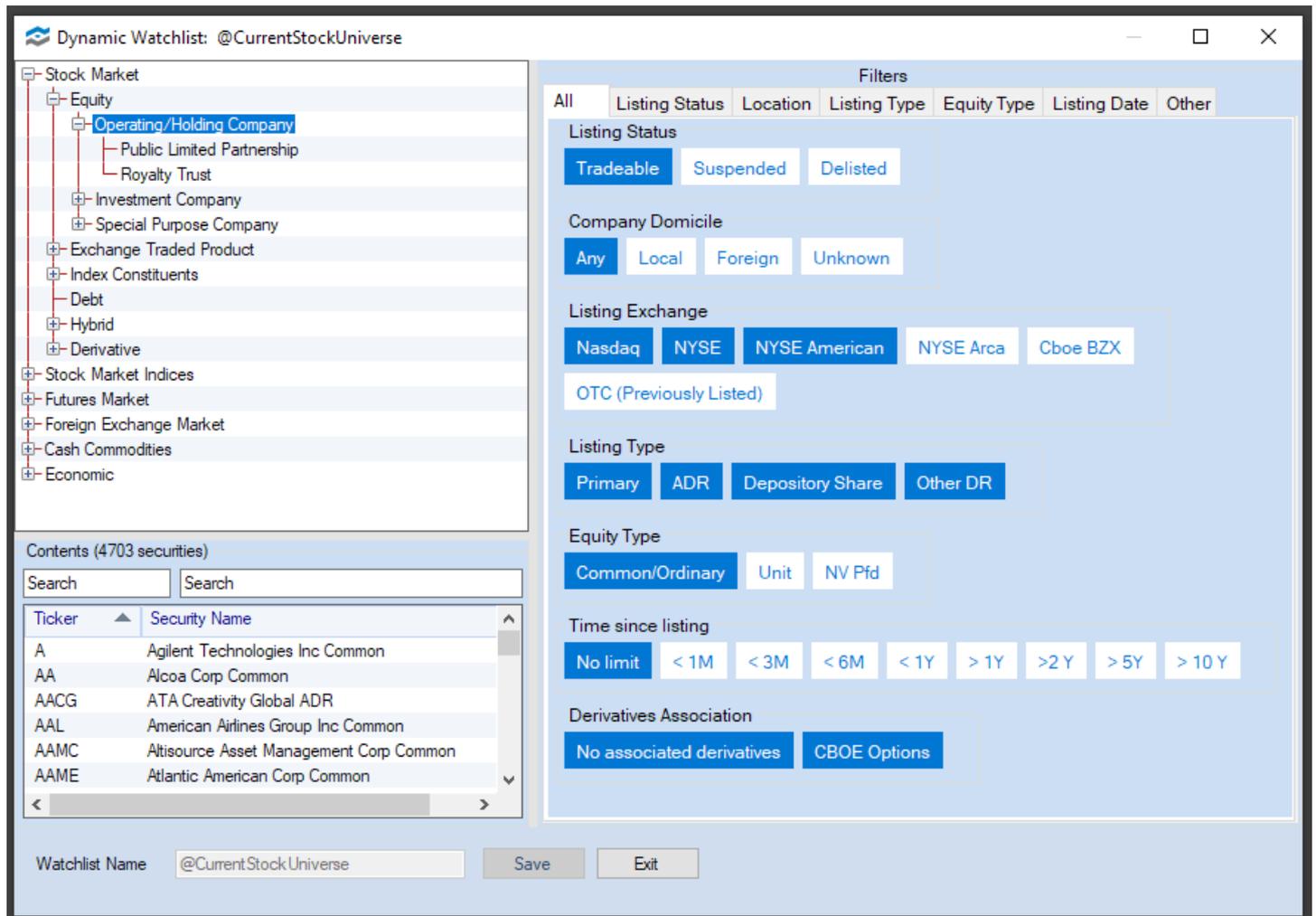
To import data from Norgate, the Norgate Data Updater (NDU) application must be currently running (open) on the same machine as RealTest.

With Norgate import, each **IncludeList** in the Import script section can be any of the following:

- One or more symbols, separated by commas
- A path to a TXT file containing a list of symbols
- A path to a CSV file in which a column of symbols has a name that contains "symbol" or "ticker" or "underlying"
- The name of a watchlist from the NDU **Watchlist Library** (prefix the name with a dot to indicate this usage)

Use of the Watchlist Library is highly recommended. You can define any number of dynamic watchlists in NDU, each of which is automatically maintained every time data is updated.

For example, I use the following Watchlist definition for my daily trading candidate scans:



This only needs to be set up once, and is referenced in my import script as follows:

```

▼ Import:
DataSource: Norgate
IncludeList: .@CurrentStockUniverse
StartDate: 1/2/2019
EndDate: Latest
SaveAs: daily_setup.rtd

```

There can be any number of include lists in an import. The first include list number that a given symbol came from can be referenced in any formula via the **ListNum** variable, and the **InList** function can be used to check whether the current symbol was in any specific list. This provides a convenient way, for example, to combine strategies that each use different and possibly overlapping sets of symbols.

Norgate makes it easy to specify the kind of data adjustment that you want. RealTest supports data adjustment specification via the **Adjustment** element of the import definition. The default and recommended adjustment to use is "Capital". Regardless of data adjustment, RealTest always models trades in a backtest (or shows data in scans) using as-traded prices, so you never have to think about split adjustment or un-adjustment when writing scripts. See **SplitHandling** for further details on this topic.

Several symbol information fields are automatically included when Norgate data is imported, specifically:

Item	Description	Content
<b>?Symbol</b>	stock symbol	string
<b>?Name</b>	company name	string
<b>?Type</b>	security type	string
<b>?Exchange</b>	exchange name	string
<b>?Domicile</b>	security country	string
<b>?Currency</b>	security currency	string
<b>?Sector</b>	security sector	string
<b>?Industry</b>	security industry	string
<b>?CII</b>	corresponding industry index	string
<b>InfoTRBC</b>	Thomson Reuters Business Classification code	<b>described here</b>
<b>InfoGICS</b>	Global Industry Classification Standard code	<b>described here</b>
<b>InfoShares</b>	number of shares outstanding	number
<b>InfoFloat</b>	number of shares float	number
<b>InfoDelist</b>	delisting date or futures expiry	date
<b>ListNum</b>	import include list number for this symbol	number
<b>PointValue</b>	point value (futures - always 1.0 for stocks)	value
<b>TickSize</b>	tick size (futures - generally 0.01 for stocks)	value

Norgate also provides a unique feature: **historical index constituency**. RealTest makes it easy to import this information with your data. Use the **Constituency** element of the import definition to specify one or more indexes that you want constituency data for (\$SPX, \$DJI, etc.) After the import, that data is available to all formulas as **InSPX, InNDX, etc.**

The **import\_norgate.rts** example script shows how this works in an import and scan:

```

Active Script - C:\REALTEST\Examples\import_norgate.rts
// shows how to import from Norgate NDU
// Norgate's standard dyanamic watchlists are used as symbol lists
// all symbols that have been in any of the 3 large-cap indexes will be included
// index constituency bits for each of these indexes will be added to every bar of every stock

▼ Import:
DataSource: Norgate
IncludeList: .Dow Jones Industrial Average Current & Past
IncludeList: .NASDAQ 100 Current & Past
IncludeList: .S&P 500 Current & Past
Constituency: $DJI,$NDX,$SPX // index constituency series --> InDJI, InNDX, InSPX
Adjustment: CapitalSpecial // use TotalReturn if you prefer dividends to appear as stock splits
Padding: None // rare to need padding, but available if you want it
Update: False // change to true to run a NDU update before the import
StartDate: 1/2/94
EndDate: 1/2/96
SaveAs: major.rtd

▼ ScanSettings:
DataFile: major.rtd
StartDate: 1/3/95
NumDays: 1

//this scan finds stocks which were members of $NDX but not of $SPX at the start of 1995

▼ Scan:
Filter: InNDX and not(InSPX)
DJI: InDJI
NDX: InNDX
SPX: InSPX

```

Date	Symbol	DJI	NDX	SPX
1/3/95	ADBE	0	1	0
1/3/95	ADCT_201012	0	1	0
1/3/95	AES	0	1	0
1/3/95	AKLMO_200808	0	1	0
1/3/95	AMAT	0	1	0
1/3/95	APCC_200702	0	1	0
1/3/95	ASCL_200504	0	1	0

In addition, Norgate offers something called "Corresponding Industry Index", which can be used to calculate, for example, the industry relative strength for any stock. The syntax used to achieve this is shown and explained in detail in the **industry\_indices.rts example script** and is used in the **cii\_rotate.rts** example.

Norgate also provides **current fundamental** data for each stock. Because these data are not historical time series, they are not useful for backtesting. Nevertheless, RealTest provides two ways to access these current fundamentals:

1. Open a **Chart** and then select *Get Information* from the **Chart Menu**. This will display all available fundamental fields by creating an HTML file and opening it in a new browser tab.
2. Request specific fundamental items to be imported by adding **Fundamentals** to your **Import** definition and specifying which items you want. Imported fundamental item values can then be accessed using **F.xxx** where xxx is the name of a specific field. **F.xxx.Date** can be used to obtain the date on which Norgate last updated that field.

## 8.2. Yahoo Import

Yahoo import works by downloading and parsing JSON data files from finance.yahoo.com. Several of the **example scripts** include Yahoo import sections, e.g.

```

Active Script - C:\REALTEST\Examples\import_spx.rts
// Import data for the current S&P 500
// components (plus SPY) from Yahoo

▼ Import:
  DataSource: Yahoo
  IncludeList: Examples\sp500.txt
  IncludeList: SPY
  SymInfoFile: Examples\spx_syminfo.csv
  StartDate: 1/2/14
  SaveAs: YahooSPX.rtd

```

To import data from Yahoo, you must provide a symbol list using one or more **IncludeList** statements. This will be the list of symbols that are downloaded and imported.

To find out what symbols are available, visit <https://finance.yahoo.com/> and use the search function, e.g.



When using Yahoo to import large lists of symbols, you may find that it becomes very slow after the first few hundred. Yahoo deliberately slows down the connection when they see too many download requests from the same IP address in too short a time. You get what you pay for (Yahoo is free).

Yahoo data includes both splits and dividends, so RealTest is able to provide accurate as-traded backtests using this data. Yahoo does not, however, offer delisted symbols or historical index constituency information. If these are important to you, **Norgate Data** is the recommended source to use.

Yahoo data also does not provide any symbol-level information such as company name or industry. Although some of that information is on the website, RealTest does not do any screen-scraping. If you have such information in CSV format and want to apply it to a Yahoo import, you can do so by adding a **SymInfoFile** path to the import definition. An example *SymInfo* file for the S&P 500 components is included in the *Examples* folder and used in the script shown above.

In addition to the advantages mentioned above, the biggest advantage to using a paid data service like Norgate has to do with this kind of *meta-data*. Specifically, there is never a need for a *SymInfo* file when using Norgate. The company name, industry, exchange, country, currency, etc. are all obtained automatically during import.

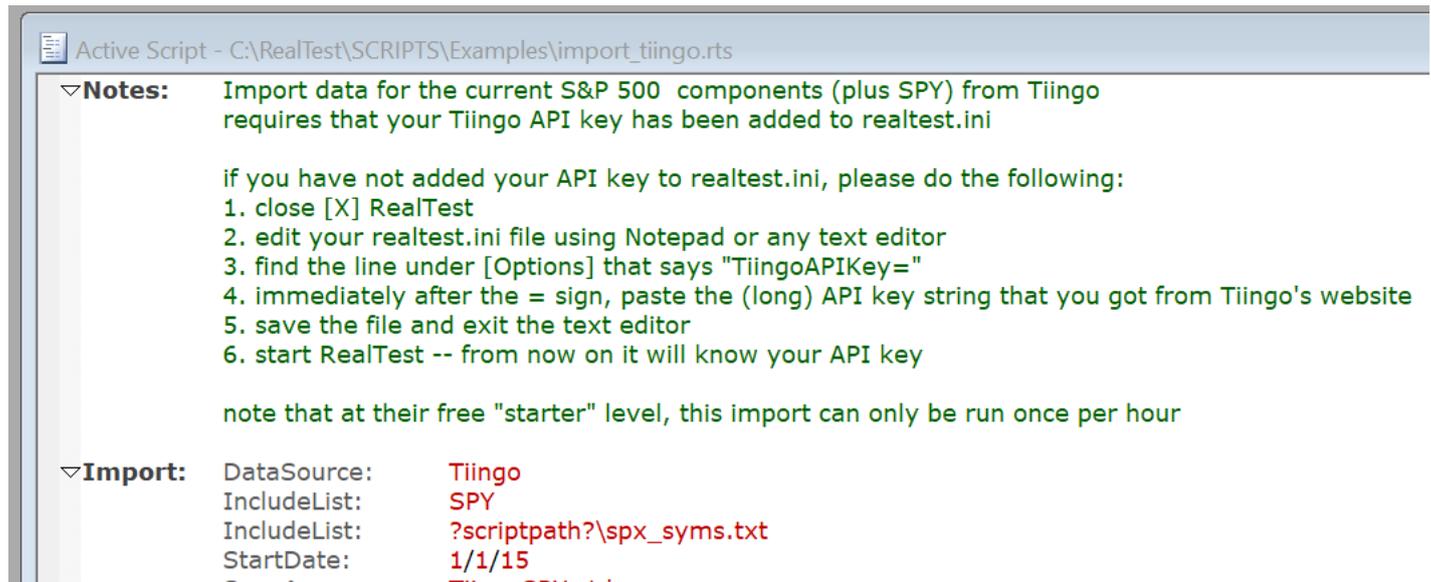
More importantly, Norgate includes a dynamic Watchlist facility that makes it easy to know that every day, you are importing "all the symbols in the market" (or in an index, or however you want to define your universe). If you use a free source like Yahoo, be prepared to do a lot of research every day just to keep track of symbol changes, additions, delistings, etc.

## 8.3. Tiingo Import

The Tiingo data service is one small step up from Yahoo in terms of quality and reliability. Like Yahoo, they offer a free service level, but limit it even more than Yahoo does, by limiting total symbols per hour to only a few hundred. For \$10/month, this restriction can be removed.

Whether you use the free or \$10/month level of service, you must register with them and receive a private API key.

The `import_tiingo.rts` example script includes comments about how to specify your Tiingo API key to RealTest in order to use Tiingo data:



```
Active Script - C:\RealTest\SCRIPTS\Examples\import_tiingo.rts
Notes: Import data for the current S&P 500 components (plus SPY) from Tiingo
requires that your Tiingo API key has been added to realtest.ini

if you have not added your API key to realtest.ini, please do the following:
1. close [X] RealTest
2. edit your realtest.ini file using Notepad or any text editor
3. find the line under [Options] that says "TiingoAPIKey="
4. immediately after the = sign, paste the (long) API key string that you got from Tiingo's website
5. save the file and exit the text editor
6. start RealTest -- from now on it will know your API key

note that at their free "starter" level, this import can only be run once per hour

Import: DataSource: Tiingo
IncludeList: SPY
IncludeList: ?scriptpath?\spx_syms.txt
StartDate: 1/1/15
CsvAs: TiingoSPY.rts
```

As stated in the above comments, you must perform a one-time task of quitting RealTest, editing RealTest.ini, and entering your key in this spot:

```
[Options]
ScriptPath=C:\RealTest\Scripts
DataPath=C:\RealTest\Data
OutputPath=C:\RealTest\Output
CalcThreads=6
MaxMemPct=95
LogDataCalcTimes=0
DateFormat=0
AskUnloadData=0
AskDeleteTests=0
AskSaveUntitled=0
CsvListView=1
ShowTextOrders=1
CheckForUpdates=1
EarliestImportYear=1990
ImportBarTime=160000
StatusBarFields=4
WriteTLDebugFile=1
TiingoApiKey=8202f68...
```

Once the INI file is saved, run RealTest and you'll be able to import from Tiingo. Your key will remain preserved in the INI file thereafter unless you delete or replace the file or edit it again and delete the key.

In addition to Tiingo stock price data, RealTest also supports *TiingoCrypto* as a separate data source. Use this DataSource name to import daily bars for any crypto currency symbol.

## 8.4. CSV Import

RealTest supports two ways to import CSV data:

1. By providing a disk folder containing multiple CSV files, one per symbol
2. By providing a single CSV file containing data for multiple dates and symbols

In either case, your CSV data files must meet the following criteria:

- Each bar of data (fields for one symbol on one date) must be on its own line (row) within each file
- The field delimiter must be a comma (unless **CSVNumFmt**: *Comma* is specified, in which case it is a semicolon)
- The decimal point must be a period (US format, again unless *CSVNumFmt: Comma* is specified)
- All rows must have the same field order

For multi-file CSV import, the following rules also apply:

- There must be one file per symbol
- The name of each file must be *SYMBOL dot CSV* (e.g. MSFT.CSV) [not case sensitive]

For single-file CSV import, the only extra rule is that the file must include a *Symbol* column.

The following potential anomalies are permitted and handled correctly in CSV import:

- Fields with or without quotes around them (quotes are removed)
- Quoted numeric fields with embedded commas (quotes and commas are removed)
- Rows in any order (ascending or descending or even random – RealTest will sort them)
- Columns in any order (but the order must be specified)
- Presence or absence of a header row or other extra rows
- Dates in any **supported format** (use **CSVDateFmt** to override current program setting for M/D/Y vs. D/M/Y if needed)
- Split adjusted and/or un-adjusted data
- Dividends can either be a column in the data, or can be specified in the EventList file as a special event type
- Each row can include an extra column with whatever you want in it

Other considerations:

- Each multi-file CSV import path must be specified in its own **DataPath** statement (there is no directory tree recursion)
- Each single-file CSV import file must be specified in its own **CSVFile** statement
- CSV import sections must also include a **CSVFields** statement to define the layout of the columns in the CSV files (header rows are always ignored)

See also **import\_csv.rts** in the **Examples** folder for an example.

## 8.5. MetaStock Import

---

RealTest supports importing data from MetaStock-format folder trees.

To import MetaStock data, specify "MetaStock" as the **DataSource**, and provide one or more **DataPath** statements with location(s) of MetaStock data folders.

Only top-level folders need to be given. RealTest will recursively traverse the entire folder tree under each specified folder.

See also **import\_ms.rts** in the **Examples** folder for an example.

## 8.6. Multi-Source Import

RealTest has the unique ability to easily combine data from multiple sources into a single .RTD data file. The **import\_multi.rts** example script shows how this works:

```
Active Script - C:\RealTest\Examples\import_multi.rts
// shows how to structure a multiple-source data import

▽ Import:
  // Each DataSource statement resets the Include/Exclude lists

  // Norgate (IncludeList(s) required)
  DataSource: Norgate
  IncludeList: &ES, SPY
  // could add more IncludeList statements if needed

  // Yahoo (IncludeList(s) required)
  DataSource: Yahoo
  IncludeList: SPY>SPY2
  // the > identifies an alias when you want to rename a symbol
  // could add more IncludeList statements if needed

  // Field layout for all subsequent CSV imports (unless restated)
  CSVFields: Date,Open,High,Low,Close,Volume,RealClose,Dividend

  // Local CSV directories where we want all the symbols
  DataSource: CSV
  DataPath: c:\NDEExport1 // include all symbols
  DataPath: c:\NDEExport2 // ditto

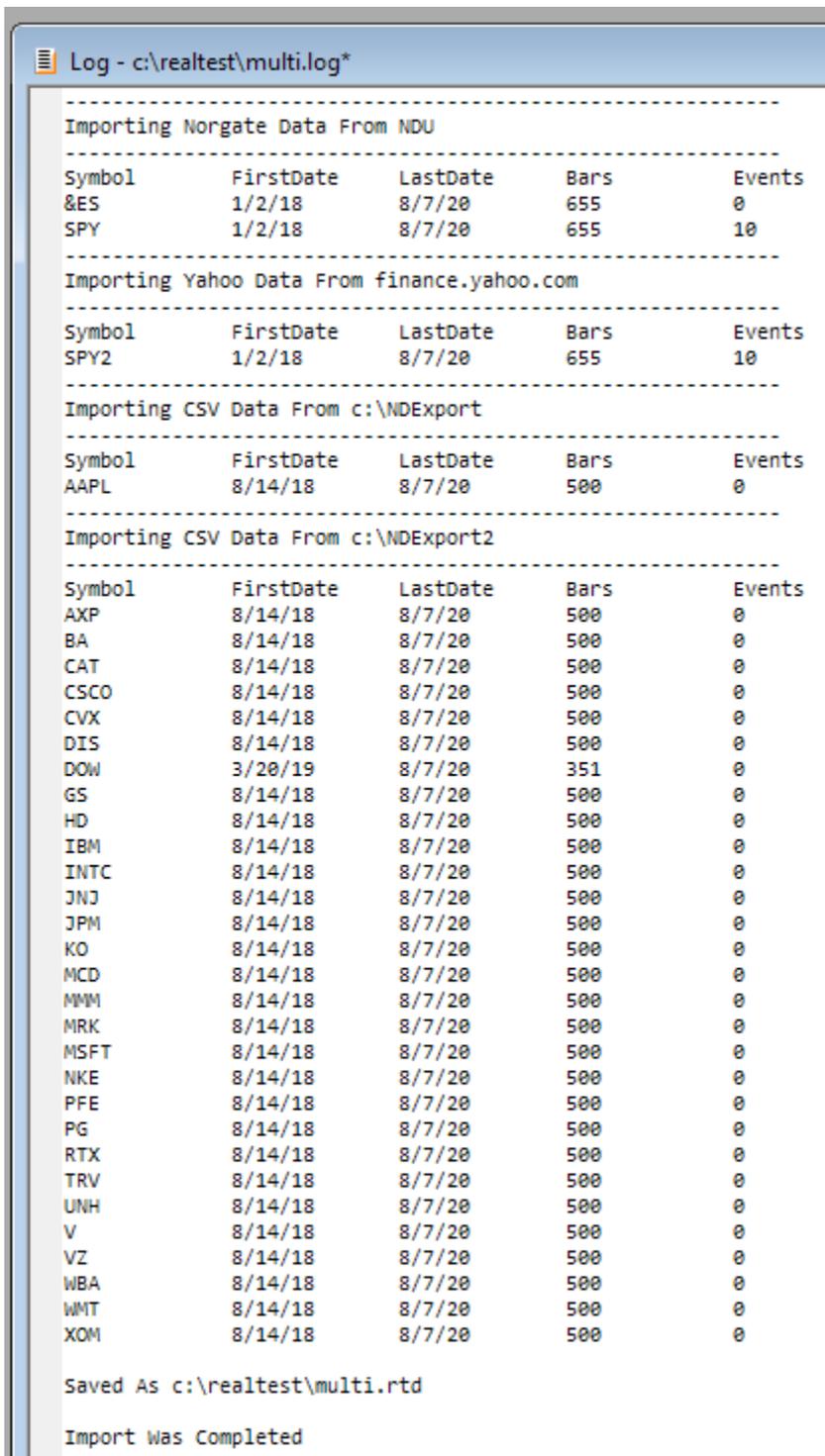
  // Local CSV directories where we want only a subset of the symbols
  DataSource: CSV // required to prevent IncludeList below from applying to DataPaths above
  IncludeList: AAPL,VZ
  DataPath: c:\NDEExport3
  DataPath: c:\NDEExport4

  // General Settings
  StartDate: 1/2/19
  EndDate: Latest
  SaveAs: multi.rtd
  LogFile: multi.log
```

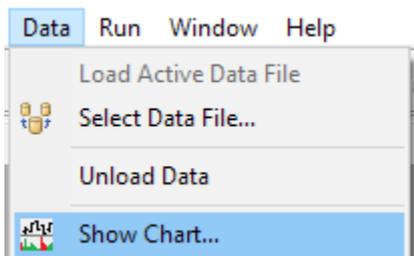
If you ran this script in Import mode (and had all the referenced data sources), the following would occur:

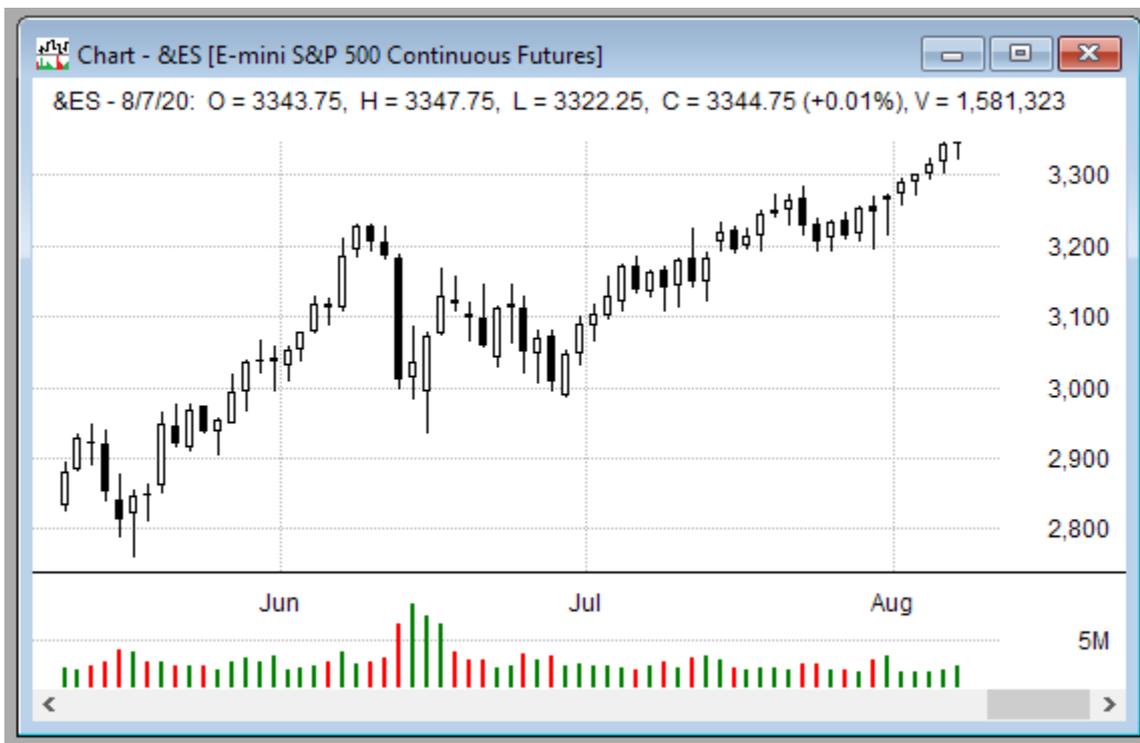
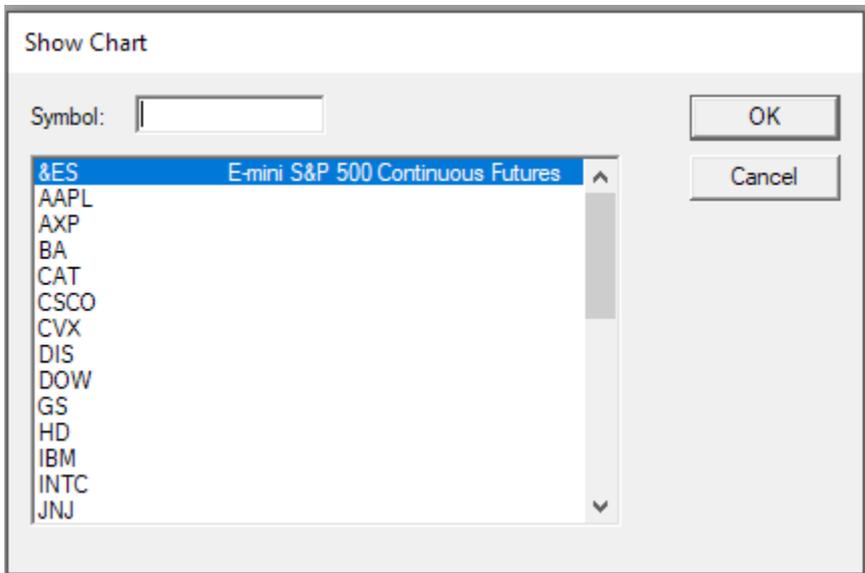
- &ES (E-Mini S&P 500 futures continuous contract) and SPY would be imported from Norgate
- SPY would also be imported from Yahoo with its symbol renamed to "SPY2"
- AAPL would be imported from a CSV data directory (assuming there is a file named AAPL.CSV in that directory)
- All symbols in a second CSV data directory would be imported (in this case the remaining DJIA components)
- All these are combined in a single RTD file (and in memory)
- The import log file *multi.log* appears

The log file confirms that all the symbols were imported:



Selecting Data / Show Chart shows that the data is now available in memory:





## 8.7. The Symbol Information File

Because some data sources provide either no information at all or only the company name, RealTest provides a way to include other per-symbol information when importing data.

This is done using a special-purpose CSV file (or multiple such files).

The first row of the file must contain column names from the first column of the following list:

Column	Content Type	Script Formula Access
Symbol	string	?Symbol

Name	string	?Name
Exchange	string	?Exchange
Currency	string	?Currency
Domicile	string	?Domicile
EconSect	string	?EconSect
Sector	string	?Sector
IndGroup	string	?IndGroup
Industry	string	?Industry
InduIndex	string	?CII
TRBC	numeric	InfoTRBC
GICS	numeric	InfoGICS
Shares	numeric	InfoShares
Float	numeric	InfoFloat
PointValue	numeric	PointValue
TickSize	numeric	TickSize
AssetId	numeric	InfoAssetId
DelistDate	date	InfoDelist or InfoExpiry

The symbol in each row should match a symbol from the import. All other columns are optional and can be in any order. Rows can also be in any order.

The set of supported symbol information columns corresponds to the metadata that is automatically included when **Norgate** is the data source. There is therefore never a reason to use a **SymInfoFile** with Norgate import.

For other data sources, you are welcome to use any of the above columns for any kind of information you want to have available. Just use the corresponding formula element to access it as needed in your scripts.

If you're using futures contract data from a source other than Norgate, it is very important that you provide a syminfo.csv file with the **PointValue** for each symbol. **TickSize** can also be provided if desired, but *PointValue* is critical for backtest stats calculations.

See also **import\_spx.rts**, **spx\_syminfo.csv**, **djia\_make\_syminfo.rts**, **djia\_use\_syminfo.rts**, and **djia\_info.csv** in the **Examples** folder.

## 8.8. Futures Symbol Information

---

For futures data import from a source other than Norgate, it is critical to include a SymInfo file with the correct point value for every symbol. Otherwise, backtest results will not be accurate. (Tick size is not required, but if present, it will be used in chart scaling, and will be useful in formulas such as slippage modeling.)

RealTest has no "futures mode" and actually doesn't know or care whether a given symbol is a stock or a futures contract. All it needs to know is the point value. This means that you can easily import data that contains a mix of stocks and futures, and test things such as using futures to hedge a stock portfolio.

There is no explicit support in RealTest for futures margin requirement levels. Use the **MaxInvested** formula as needed to model your margin needs. If you need to refer to per-contract margin requirements in your strategies, you can use one of the extra Info columns in the SymInfo file to import this data if available.

## 8.9. The Event List File

---

Event List files can be included in a data import to add point-in-time event information to specific dates for specific symbols. The most likely use of this feature would be to include historical earnings dates and

values if you have access to such information.

As with the symbol information file, the event list is a CSV file (or group of files).

The first row of the file must contain column names from the following list:

- Symbol – the symbol for which the event occurred
- Date – the date of the event
- Time – the time of the event (optional)
- Type – any numeric code > 0
- Value – any numeric value (e.g. dividend amount, or EPS, or index constituency flags)

These columns can be in any order. The names are used to identify them.

Rows after the header row can also be in any order. Each row defines one event for one symbol on one date.

When first importing with an event list, it is suggested to use the **LogFile** option in your import definition. Events for which the symbol+date is not found in the imported bar data will be listed in the log.

Imported event data is accessed in script formulas by using the **Event(type)** function, which returns the value field if an event exists for the current symbol on the current date, or 0 if no event was found (with an option to return the most recent event -- see the function link for details).

The *Time* field stored with each event record determines which bar the Event function will align the event with. If  $Time \leq 16:00:00$  (160000) then the event will be aligned with the bar which has the specified date. If  $Time > 160000$  then the event will be aligned with the next bar after the bar with the specified date.

The intention with this distinction is to differentiate "before the open" vs. "after the close" earnings report times. If you have point-in-time earnings data and are using **EventListFile** to import it and *Event(n)* to reference it, you will want *Event(n)* to return a value only for the dates on which the market first reacted to earnings news.

See also **djia\_earnings.rts** and **djia\_earnings.csv** in the **Examples** folder (note that this example does not use the *Time* field).

The following is not necessary to read or understand unless you have your own custom dividend and/or index constituency data series that you need to add to a non-Norgate import.

As well as for user-defined events as described above, RealTest also uses the event list within each RTD file to store **Dividends** and **Constituency Changes** events.

Dividends are stored as event type -1. *Event(-1)* is therefore equivalent to *Dividend* in any formula. *Event(-1, 1)* can be used if desired to find the most recent dividend amount. *Event(-1, 1)* is equivalent to *WhenTrue(Dividend > 0, Dividend)*.

To add your own dividend event, set Date to the ex-dividend date, Type to -1, and Value to the dollars-per-share amount of that dividend on that date.

Constituency Changes are stored as event type -2 through -n. The specific event number for an index is -1 minus the number from the first column of the *constituency.csv* file. So by default \$MEL is -2, \$XAO is -3, etc.

RealTest automatically generates a constituency change event on the first available date of each symbol in the RTD file for each imported constituency series (index) that it belonged to on that date.

Though constituency change events are only present for the dates (after the first date) on which the symbol entered or left the index, RealTest automatically maps the corresponding **InXXX** variable to *Event(n, 1)*, i.e., it looks up the most recent change event to determine whether the stock was in the index on the date being evaluated.

To add your own index constituency series, do the following:

1. Edit your copy of the *constituency.csv* file and add your indices at the end. Perhaps start with 100 is the first index number, to leave plenty of space after the Norgate numbers. Make up your own values for the remaining columns, e.g. **100, \$DAX, InDAX, Member of Dax**. The only required items here

are the number and the InXXX name to use -- the other two are provided only for your own information since you would not be directly importing these series from Norgate.

2. Create an EventListFile containing your constituency change events. Symbol is the stock to which the change applies. Date is the date of change. Type is -1 minus your index number as specified in step 1 above, e.g. -101 for index 100. Value is 1 if the stock joined the index or 0 if the stock left the index on that date.

3. Close and re-open RealTest so it can read your new constituency.csv file

4. Run your import

You should now be able to reference your custom index constituency series using your custom InXXX variable!

## 9. Running Scans

---

Scans are used when we want to extract a subset of data from the currently loaded data file and display that data in a specific way.

If you've gone through **Tutorial 3**, then you already know a lot about Scans.

To run a scan, you must first import or load a data file, then open or create a script that includes a **Scan** section.

If the currently active script includes a Scan section, then the *Scan* button on the Tool Bar (and the *Scan* item on the **Run Menu**) will be enabled. Pressing it will run the script in scan mode.

As with all **run modes**, *Scan* first applies your settings from the **Settings Panel**, then applies the **Settings** script section if one is present. Finally, if your script includes a **Scan Settings** section, items that it specifies will override your general-purpose settings for those items.

The *Filter* formula of a scan defines which symbols and bars are included in the output.

The *Sort* specification item lets you define the initial sort order by column name.

All other items in a scan definition are used to add columns to the output with any desired contents.

For some examples of how to use scans to generate candidate or order lists for daily trading, see **Daily Setups Scan**, **Multi-Row Scan** and **Test Output Scan**.

# 10. Running Tests

To run a test, you must first import or load a data file, then open or create a script that includes one or more **Strategy** sections.

If the currently active script includes a *Strategy* section, then the *Test* button on the Tool Bar (or the *Test* item on the **Run Menu**) will be enabled. Pressing it will run the script in backtest mode.

As with all **run modes**, *Test* first applies your settings from the **Settings Panel**, then applies the **Settings** script section if one is present. Finally, if your script includes a **Test Settings** section, items that it specifies will override your general-purpose settings for those items.

If you haven't already done so, please go through **Tutorial 1** and perhaps the others to get a feel for how backtesting works in RealTest.

For a look "under the hood", see **Backtest Engine Details**.

## 10.1. Multiple Tests and Optimization

When you run a test in Optimize mode, the optimization dialog appears before the test starts running.

The screenshot shows the 'RealTest Optimizer' dialog box. It features a table for parameter selection, several radio button options for optimization modes, and various input fields for test parameters. Numbered callouts (1-9) point to specific elements: 1. Parameter List table; 2. Optimization Mode section; 3. For Each Symbol checkbox; 4. Test Iterations spinner; 5. Score dropdown; 6. Results Window Options section; 7. Date Intervals section; 8. Create Walk-Forward checkbox; 9. Maximum Test Count spinner.

Parameter	Count	Values (first is default)
<input checked="" type="checkbox"/> MA1	11	5, 6, ..., 14, 15
<input checked="" type="checkbox"/> MA2	10	10, 20, ..., 90, 100

- 1 Parameter List**

Shows each of your optimization variables along with the count of iterations for that variable and the range of values. The checkboxes within the list can be used to exclude any variable from optimization. When a variable is excluded, its default value will be used.

2

## Optimization Mode

Specifies the type of optimization to be performed.

- **Combinatorial** mode is the traditional exhaustive nested loop covering every possible parameter combination. If you have more than 2 or 3 variables with more than a few values for each, you will see that the total test count quickly becomes very large. Due to the speed of RealTest, it's quite practical to run portfolio-level optimizations involving thousands of tests across thousands of stocks across multiple years of time. Whether this approach is likely to discover a strategy that will be profitable in the future is another matter.
- **Sequential** mode loops over each variable in order. At the end of a loop, that variable keeps whichever value produced the highest score. Running 2 or 3 iterations of a sequential optimization will often be a quicker way to find areas of good parameter values than the combinatorial approach.
- **Genetic** mode is not a true genetic optimizer, but the concept is similar. Before each test, a random subset of the parameters is selected and then the values of those variables are selected at random from their value lists. (How many parameters to change each time is determined by the "mutate" option.) If the score from the test is higher than the prior best score, the new values are kept, otherwise the prior best values are restored. Genetic mode will generally converge on the best combinatorial result within approximately the square root of the total combination count. Note that to use genetic mode, you must specify the number of test iterations to be run.
- **Random** mode is similar to genetic mode except that it randomly selects a value for every variable before every test, and (therefore) completely ignores the score of each test. A good use of random mode is to select a reasonable value ranges for each parameter, run 100 test iterations (random combinations), and look at the median result. This might provide a reasonable estimate of how the system would perform in the future, given how arbitrary parameter selection can be. Note that to use random mode, you must specify the number of test iterations to be run.

3

## For Each Symbol

This option can be used whether or not the script contains any parameters. If checked, rather than running a single test (or multi-test optimization) which scans all symbols each day, RealTest will run a separate test (or a separate multi-test optimization) for each symbol in the data file. This makes it easy, for example, to find the best parameters for each symbol and then look at the combined results to get a feel for the robustness of those parameters. (For a per-symbol optimization, the settings Sort After Each Test, Keep Best 1, and group By Symbol are recommended.)

4

## Test Iterations

Use this setting to do any of the following:

- run many iterations of the same test repeatedly using the **Random** function in one or more strategy formulas, to obtain a range of potential outcomes
- run a parameter optimization in **Genetic** mode, to specify how many iterations to run
- run a parameter non-optimization in **Random** mode, to specify how many iterations to run

5

## Score

Allows you to select any column from your results window to use as your "fitness function". Since the results columns are all formula-based, this means you have unlimited possibilities for what to use here. The value returned by the selected formula is used to rank the results after an optimization run if "Sort After Each Test" is checked. This is most useful with the Sequential or Genetic optimization modes, and when creating a Walk-Forward test.

6

## Results Window Options

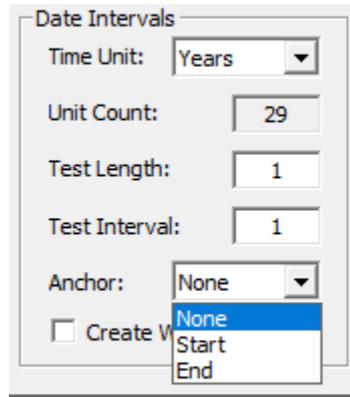
The results window options choices enable you to clear the results window at the start of the test (assuming it was already open, otherwise a new one is created), and optionally be partially cleared and sorted during the test as well. While watching an optimization run, it is often desirable to just see the best 10 results, for example.

7

## Date Intervals

The Date Intervals panel is used primarily to generate a **walk-forward** test but can also be used to simply produce a series of results for different time periods. (It is OK to run a script in Optimize mode even if it contains no optimization parameters.) In particular, it can often be useful to set the time unit to "Years", simply to run a strategy for each year of a date range separately.

By default each date interval is a sliding window of the same length. The *Anchor* setting lets you optionally lock the start or end date.



Anchoring the start date makes each interval larger than the preceding one, and anchoring the end date does the opposite.

8

## Create Walk-Forward

See **Walk-Forward Tests** for a detailed description of this feature.

9

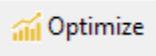
## Maximum Test Count

At the lower-right corner of the dialog is the test count. Notice how it changes as you check and uncheck the checkboxes.

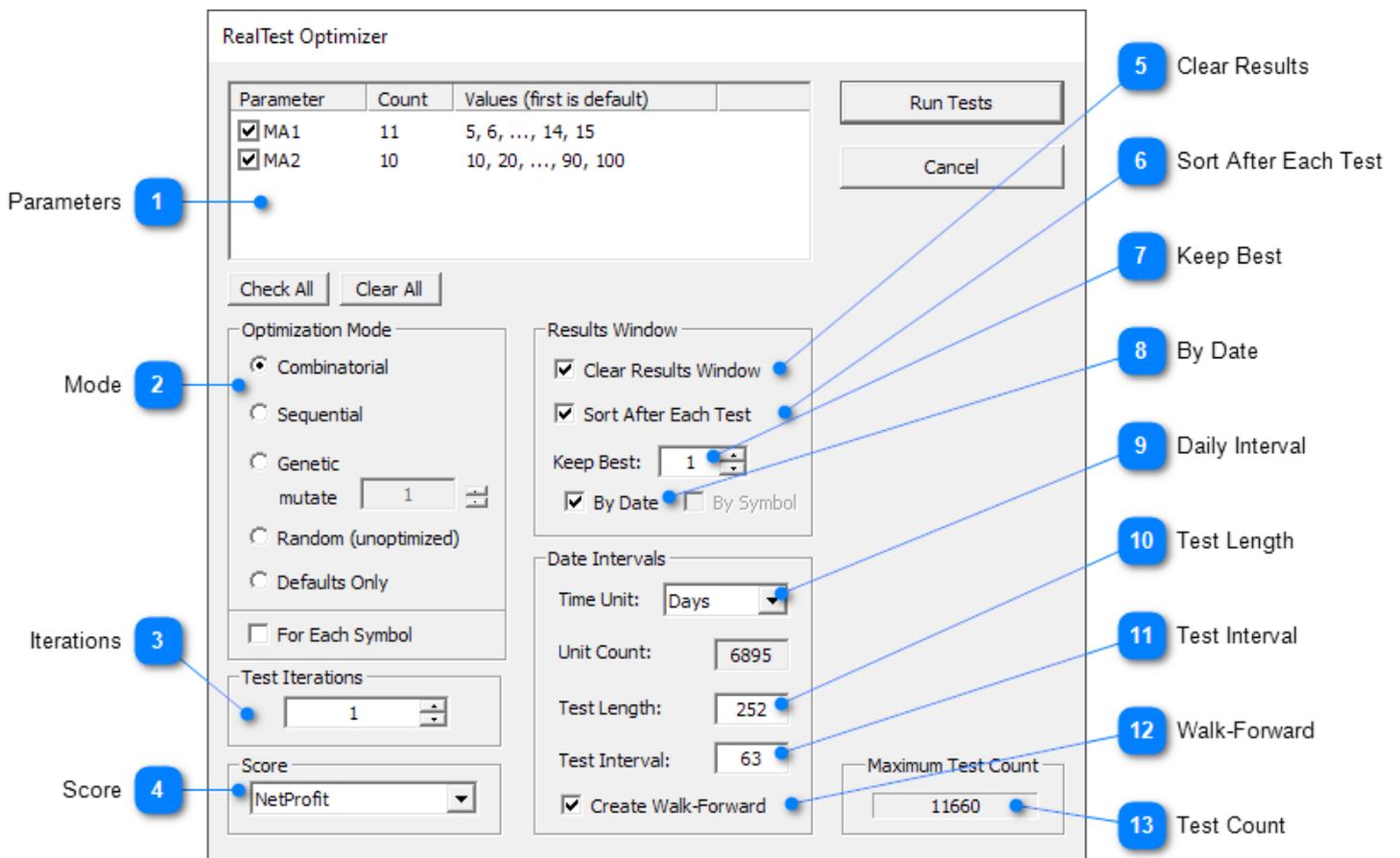
See also **Tutorial 2 - Optimization** for a specific usage example.

## 10.2. Walk-Forward Tests

To introduce Walk Forward testing, let's return to the sample2.rts script used in **Tutorial 2**.

Open that script again and click on  **Optimize** .

Now make all of the following selections in the Optimizer dialog:



- 
- 1 Parameters**  
Check both parameters.
- 
- 2 Mode**  
Set to "Combinatorial"
- 
- 3 Iterations**  
Set to 1.
- 
- 4 Score**  
Set to NetProfit, or any other Results Window item (statistical formula) you want to optimize for. This is your "Fitness Function".
- 
- 5 Clear Results**  
Check this so any prior tests will be deleted, and/or open a new results window.
- 
- 6 Sort After Each Test**  
Check this whenever generating a Walk-Forward test.
- 
- 7 Keep Best**  
Set to 1 for Walk-Forward.
- 
- 8 By Date**

---

Check this (it will become enabled after the next step if it isn't already).

---

9

### Daily Interval

---

Set Time Unit to Days (or a longer unit if desired).

---

10

### Test Length

---

Specifies how many days (since that's the Time Unit) to include in each backtest (252 days is one year).

---

11

### Test Interval

---

Specifies how often to re-optimize (63 days is one quarter).

---

12

### Walk-Forward

---

Check this.

---

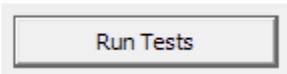
13

### Test Count

---

This will show how many tests need to be run.

---

Now you are ready to press  and let RealTest generate your walk-forward model.

Though there are more than 10,000 tests to run, it will probably take less than one minute to finish, depending on the speed of your computer.

As the tests run, you will see the Results window being updated continually. To help the process finish faster, minimize the Results window until all tests are finished, then restore it again.

Here is what is going on under the hood:

*for each 63-bar interval of your overall date range*

*for each of the 110 possible combinations of your two parameters*

*run a 252-bar test and keep the result if it is better than the prior best result*

*record the start date and parameter values that produced the best result*

Once the above is complete, a new **WalkForward** section is automatically added to your script. This section contains an item called "Dates", and an item for each of your parameters. These things together serve to define how to run the final Walk-Forward test, in which the best parameters from the *previous* interval are used for each 63-bar interval. In effect, this becomes a test of the effectiveness of the process of periodic re-optimization.

Here is a snippet of how the results and script windows will look at the end of the run:

Test	Notes	Dates	MA1	MA2	Periods	NetProfit	ROR	MaxDD
0794	Walk-Forward	1/28/94 - 5/7/20	15	100	6,615	\$264,931	5.06%	-30.11%
0793	Sample2	5/9/19 - 5/7/20	5	20	252	\$24,936	24.94%	-7.58%
0789	Sample2	2/7/19 - 2/6/20	5	20	252	\$21,250	21.25%	-3.20%
0786	Sample2	11/5/18 - 11/5/19	7	10	252	\$13,470	13.47%	-6.85%
0784								-6.09%
0777								-5.84%
0772								-13.07%
0767								-8.91%
0763								-10.10%
0756								-10.09%
0746								-6.99%
0739								-3.03%
0724								-1.92%
0719								-5.31%
0715								-5.31%
0711								-6.82%
0708								-8.30%
0706								-8.82%
0704								-8.83%

Active Script - C:\REALTEST\Examples\Sample2.rts

```
// 2-Parameter optimization example
// find out which crossover MA lengths would have done best

TestSettings:
  DataFile: spy.rtd
  StartDate: Earliest
  EndDate: Latest

Parameters:
  MA1: from 5 to 15
  MA2: from 10 to 100 step 10

Strategy: SPY_Crossover
  EntrySetup: Avg(C,MA1) > Avg(C,MA2)
  ExitRule: Avg(C,MA1) < Avg(C,MA2)

WalkForward: // unit=Days, length=252, offset=63
  Dates: 1/28/94, 5/2/94, 8/1/94, 10/28/94, 1/30/95, 5/1/95, 7/31/95, 10/
  MA1: 11, 11, 15, 15, 11, 13, 10, 6, 10, 15, 5, 11, 11, 15, 10, 12, 11, 10,
  MA2: 10, 30, 30, 30, 10, 30, 20, 20, 60, 70, 80, 10, 10, 40, 40, 20, 20, 20

// each of the above 3 items continues far to the right...
```

To run the walk-forward test again (without re-generating all the parameter values), simply press the **Test** button or select *Run Test* from the **Run Menu**:

When a script includes a *WalkForward* section (and it's not commented out), RealTest will always use it when running that script as a single backtest.

Now for (an unfair) comparison, let's do the following:

1. Change the date range in Settings to match the dates of the Walk-Forward row in your results window.

In my case this will be:

```
TestSettings:
  DataFile: spy.rtd
  StartDate: 1/28/94
  EndDate: 5/7/20
```

(The start date is 252 bars after the earliest data date, and the end date is 252 bars after the last multiple of 63 bars that would not push it past the last data date.)

2. Add default values for each of the two parameters to select 10 and 70, which were among the best results found in the overall optimization done in Tutorial 2.

```
Parameters:
  MA1: from 5 to 15 def 10
  MA2: from 10 to 100 step 10 def 70
```

3. Comment out the *WalkForward* section, so that the test will run with constant parameter values.

4. Run a regular single test by pressing  .

5. Look at the new line at the top of the results window:

Test	Notes	Dates	MA1	MA2	Periods	NetProfit	ROR	MaxDD
0796	Sample2	1/28/94 - 5/7/20	10	70	6,615	\$481,515	6.94%	-31.53%
0794	Walk-Forward	1/28/94 - 5/7/20	15	100	6,615	\$264,931	5.06%	-30.11%

(Ignore the parameter values for the Walk-Forward line - they should really be blank or say "various".)

In this case it appears that the walk-forward process produced results that were roughly equivalent to the most optimal overall result.

See also *spy\_tlt\_uis.rts* in the *Examples* folder for an example that uses walk-forward optimization to implement a strategy from an article.

# 11. Using an Imported Trade List

RealTest makes it easy to define a strategy based on a list of specific trades.

Running such a strategy is like playing back those trades and using them to generate results statistics.

Reasons to do this might include:

- **Analyzing** the results from a period of live trading (whether systematic or discretionary)
- Seeing if you could have done better using different **exit rules, position sizing, entry skip** criteria, etc.
- Comparing actual vs. backtest results for a strategy or set of strategies
- Playing back live trades to correctly establish current positions when generating **Tomorrow's Orders**.

RealTest supports two kinds of trade lists: *transactions* and *round-trip trades*.

Transaction lists are things like IB Flex Query output, where each row only specifies a single transaction (e.g. BUY 100 MSFT).

RealTest automatically sorts the transaction list by symbol then date and time, and converts it to a round-trip trade list for use in a backtest.

Round-trip trade lists include one row per round-trip trade, with both the entry and exit details included.

In either case, the trade list file must be in a standard CSV format, where the field delimiters are commas and decimal points are periods.

To use a trade lists of either kind, you must provide a definition of what is in each column.

See **TLFields** for details on what columns can be used and how to specify their layout. (The header row of the CSV file is ignored and is not required.)

See **TLAdjusted** and **TLTimeShift** for other settings related to trade lists.

As an example, see **actual\_trades.csv** in the Examples folder, the first few rows of which look like this in a text editor:

```
Log - C:\REALTEST\Examples\actual_trades.csv
Symbol,Strategy,Side,DateIn,QtyIn,PriceIn,DateOut,QtyOut,PriceOut
LITE,Longs,1,1/3/2017,269,37.259206,1/4/2017,269,37.5
LIVE,Longs,1,1/6/2017,565,17.787138,1/9/2017,565,18.883708
PI,Longs,1,1/5/2017,317,31.839957,1/10/2017,317,34.849297
RRGB,Longs,1,1/6/2017,200,50.495,1/10/2017,200,49.307955
SCHN,Longs,1,1/6/2017,423,23.8,1/10/2017,423,25.051299
MTZ,Longs,1,1/9/2017,292,34.441874,1/10/2017,292,35.7
PI,Longs,1,1/18/2017,348,29.543699,1/20/2017,348,31.07
HP,Longs,1,1/27/2017,145,72.133711,2/1/2017,145,72
MUR,Longs,1,1/30/2017,366,28.476043,2/1/2017,366,29.09
IP,Longs,1,2/6/2017,206,51.224229,2/7/2017,206,52.139621
```

or like this in Excel:

	A	B	C	D	E	F	G	H	I
1	Symbol	Strategy	Side	DateIn	QtyIn	PriceIn	DateOut	QtyOut	PriceOut
2	LITE	Longs	1	1/3/2017	269	37.25921	1/4/2017	269	37.5
3	LIVE	Longs	1	1/6/2017	565	17.78714	1/9/2017	565	18.88371
4	PI	Longs	1	1/5/2017	317	31.83996	1/10/2017	317	34.8493
5	RRGB	Longs	1	1/6/2017	200	50.495	1/10/2017	200	49.30796
6	SCHN	Longs	1	1/6/2017	423	23.8	1/10/2017	423	25.0513
7	MTZ	Longs	1	1/9/2017	292	34.44187	1/10/2017	292	35.7
8	PI	Longs	1	1/18/2017	348	29.5437	1/20/2017	348	31.07
9	HP	Longs	1	1/27/2017	145	72.13371	2/1/2017	145	72
10	MUR	Longs	1	1/30/2017	366	28.47604	2/1/2017	366	29.09
11	IP	Longs	1	2/6/2017	206	51.22423	2/7/2017	206	52.13962

A trade list file can include all of the trades or transactions from any number of strategies.

The example script **actual\_trades.rts** shows how to use this file.

This example includes two strategies, called "Longs" and "Shorts" (the above screen shots happen to only include long trades).

To "run" this trade list as a backtest that produces separate stats for each side plus combined stats, the example defines two strategies:

```
▽ Template: trades  
TradeList: Examples\actual_trades.csv  
TLFields: Symbol,Strategy,Side,DateIn,QtyIn,PriceIn,DateOut,QtyOut,PriceOut  
  
▽ Strategy: longs // must exactly match a name found in the strategy column in the trade list  
Using: trades  
  
▽ Strategy: shorts // ditto  
Using: trades  
  
// strategy sides above are inferred from tradelist side column -- their use in strategy names is just a coincidence
```

The **Template** mechanism was used to avoid having to repeat the file path and field map.

This example script contains sample templates for several known trade and transaction list formats, which can be copied in to your own scripts as needed:

```
// this works for IB TWS trade log export files with default columns  
  
▽ Template: iblog  
TradeList: ib_log.csv // change to your actual file (assumes Order Ref contains strategy name)  
TLFields: Symbol,Action,QtyIn,PriceIn,TimeIn,DateIn,,,strategy  
  
// this is one example of a IB flex query with too many columns selected  
// it is recommended to configure your flex query with only the columns that you need for this purpose  
  
▽ Template: flex  
TradeList: ib_flex_query_output.csv // change to your actual file  
TLFields: ,,Symbol,,,,,,,,Underlying,,,,,,,,,,,,,DateTime,,,,,Action,QtyIn,PriceIn,,,FeesIn  
  
// this is for IB flex query for ASX trades, where the Date/Time column is in NYC time  
// the TLTimeShift setting tells RT how to shift the date/time values so they're aligned with ASX data bars  
  
▽ Template: asxflex  
TradeList: asxflex.csv  
TLFields: DateTime,Action,Symbol,QtyIn,PriceIn,FeesIn  
TLTimeShift: 14 // NYC->Sydney  
  
// this is for a standard AmiBroker trade list (run test with trades report then export to CSV)  
// AB trade prices are split-adjusted, so use TLAdjusted to tell RT to unadjust them  
  
▽ Template: abtrades  
TradeList: abtrades.csv  
TLAdjusted: True  
TLFields: Symbol,strategy,DateIn,PriceIn,DateOut,PriceOut,,,,Shares  
  
// this is for TradeRunner's trades.csv output file  
  
▽ Template: tradelist  
TradeList: c:\traderunner\trades.csv  
TLFields: Symbol,strategy,side,,DateIn,,,,QtyIn,PriceIn,FeesIn,,,,DateOut,,,,QtyOut,PriceOut,FeesOut
```

## Overlaying Trading Rules

In addition to simply "backtesting" a list of trades to generate stats and graphs or look at charts, you can also experiment with adding strategy elements to override the entry or exit rules or position sizes of the trades.

As a simple example, try adding "EntrySkip: C > 50" to the strategies in this example script, then re-running it. You'll see that this reduces the trade count by about half, and lowers the expectancy somewhat.

To override the position sizes in an imported trade list, add both a **Side** specification and a **Quantity** formula to each strategy.

To test different exit rules, simply add them to the strategy.

Note that adding any exit formula (ExitRule, ExitLimit or ExitStop) to a TradeList strategy disables the exits from the trade list.

### Using Hybrid TradeList+Formula Strategies for Tomorrow's Orders

As described in the prior section, any of the standard **Strategy Elements** can be added to a *TradeList* strategy to override certain details of the trades in the list.

A special exception to the usual way this works occurs when a test is run in **Orders Mode**. In this case, the presence of *TradeList* tells RealTest that you want to use this special hybrid mode designed for live trading.

In this special mode, the trade list is played back "as is", ignoring all other strategy elements, up to and including the last date of the test.

At that point, using the current equity value and open positions from the trade list playback, the strategy formulas are evaluated for the upcoming day. This produces an accurate set of orders and quantities for tomorrow based on your actual trades and current positions.

To use this feature in your daily live trading, you would need to do the following:

1. Obtain an accurate record of your live trades, in CSV format, since the desired start date. Ideally, your automation software will generate this list automatically. Alternatively, it could be produced by IB using either Flex Query (via the account management website) or by enabling automatic generation of trade logs in TWS and then concatenating them.
2. Add *TradeList* and *TLFields* statements to your existing strategy definition, referencing the CSV file thus obtained.
3. Import the latest data and run the test in this hybrid mode to produce tomorrow's orders.

Note that when **OrderClerk** is used to place and manage your orders, all of the above is handled automatically. *OrderClerk* maintains an accurate round-trip trade list from your live execution reports. By specifying **OrdersMode: OrderClerk** and **OrderClerkFolder: <path to folder where OrderClerk runs this strategy>**, you don't have to add *TradeList* or *TLFields* to the strategy and don't need to worry about how to maintain a live trade list.

## 12. Using Command Line Mode

---

RealTest supports an optional Windows command line mode. To use this mode, simply invoke RealTest from any Windows command shell window or batch script, with arguments specifying your desired action.

Supported command line tasks:

❖ **realtest -import *script.rts***

- runs *script.rts* in **Import** mode
- saves data to RTD file if import definition includes a **SaveAs** path

❖ **realtest -scan *script.rts***

- runs *script.rts* in **Scan** mode
- saves scan output to CSV if **ScanSettings** include a **SaveScanAs** path

❖ **realtest -test *script.rts***

- runs *script.rts* in **Test** mode
- saves results data to RTR if the **Settings** include a **ResultsFile** path
- saves results window contents to CSV if test settings include a **SaveTestListAs** path
- saves stats to CSV if test settings include a **SaveStatsAs** path
- saves final position list to CSV if test settings include a **SavePositionsAs** path
- saves trade list to CSV if test settings include a **SaveTradesAs** path

❖ **realtest -orders *script.rts***

- runs *script.rts* in **Orders** mode
- generates and saves Order List file(s) as specified in the script
- also saves results and other output as is done for *Test* mode

Multiple run modes can precede a script name, e.g. **realtest -import -test *script.rts*** will run that script first for import and then for backtest.

Multiple scripts can follow a run mode, e.g. **realtest -orders *orders1.rts orders2.rts orders3.rts*** will generate orders from three different scripts.

If all scripts ran successfully, the program return code will be 0.

If there were errors, the return code will be one of the following:

Code	Meaning
1	unknown command line task
2	license error
3	file open/read/write error
4	memory error
5	script error (syntax etc.)
6	import error
7	can't load data file
8	other / unspecified

Note that in the event of an error, the usual popup message that would be displayed when not running in command line mode is instead appended to a file called "errorlog.txt" in the folder where RealTest was installed.

When using a standard Windows .CMD batch script, you can check the return code using %errorlevel%, as in the following example:

```
@echo off
cd c:\RealTest

start /wait RealTest -import -scan my_scan.rts
if errorlevel 1 goto error

start /wait RealTest -import -orders my_orders.rts
if errorlevel 1 goto error

echo Script completed
goto done

:error
echo An error occurred -- see Notepad for details
start notepad c:\RealTest\errorlog.txt

:done
pause
exit
```

Note also that in batch mode, RealTest runs with its main window minimized, but you can restore it manually if desired, to see what it's doing.

The command **-noexit** can optionally be added to the command line. If found, RealTest will remain open after processing the other commands.

If errors are encountered in a -noexit run, RealTest will close, return an error code, and log the error to errorlog.txt, as if -noexit had not been specified.

# 13. Using Multiple Instances

---

You are welcome to run any number of simultaneous instances of RealTest on the machine(s) for which you have activated your license.

Each instance everything that it needs to run scripts in its own allocated memory, so they will not collide or conflict with each other in any way.

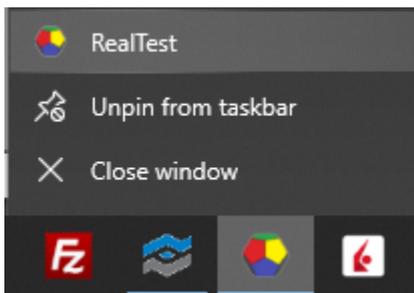
The only possible exceptions to this are:

1. The RealTest.ini file, which is used to remember things like window positions, default settings and recent file lists, is automatically saved when RealTest is closed, so whichever instance is closed last will be the one whose settings etc. are saved in RealTest.ini.
2. Similarly, if multiple instances are running the same script file, the file saved to disk will be the one from whichever instance has run it most recently. (Script changes are always automatically saved before a script is run.)

You can optionally copy your entire RealTest folder to one or more other locations on your disk drive, in order to avoid both of the above potential conflicts and maintain completely separate work environments. In most cases this will not be necessary, but it is an option if you need it.

To run multiple instances of RealTest from its desktop icon, simply double-click on the icon multiple times.

To run multiple instances of RealTest from its task bar icon, right-click on the task bar icon and select RealTest from the popup menu.



This starts a new instance, whereas clicking on the taskbar icon just brings the current one to the front (or pushes it to the back if it's already in front).

Another way to start new instances is to open a command prompt, navigate to the RealTest folder (e.g. "cd c:\RealTest"), then type "RealTest" and hit enter.

---

RealTest can optionally be told to use a file other than RealTest.ini to retrieve and store program settings etc.

This is done using the **-inifile** command line option.

For example `RealTest -inifile project1.ini` would run RealTest using that settings file rather than the default `RealTest.ini`.

This use of multiple ini files can help avoid the potential confusion of two instances both updating the same ini file.

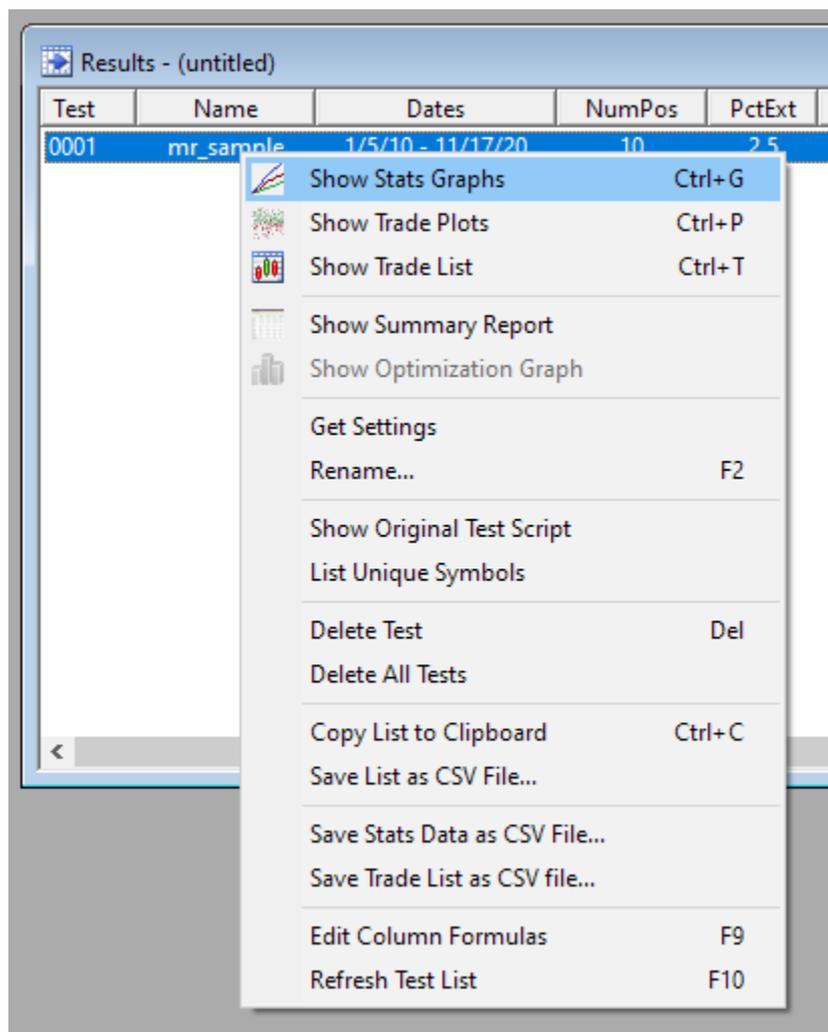
It can also provide a way to maintain separate settings, open and recent file lists, etc. for different projects.

# 14. Analyzing Test Results

After **running a test**, a new **Results** window will appear or, if one was already open, a new row will appear in it.

Each test that has been run gets its own row in a results window.

Behind each of these rows is a richness of information:



Here are links to details about many of the above items:

- [Daily Stats Graph](#)
- [Trade Plot](#)
- [Trade List](#)
- [Summary Report](#)
- [Optimization Graph](#)

**Show Original Test Script** will open a new read-only script window displaying the exact script that was used to run this test. This is especially useful while repeatedly editing and re-running a script, when you can't quite remember how a particular result was achieved.

**Get Settings** resets the **Settings Panel** to show the settings that were used when this test was run.

**Modify Notes** allows you to edit the value of the "Notes" column for this test ("mr\_sample" in the above image)

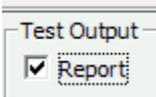
**List Unique Symbols** creates a new log window with a sorted list of each symbol that was traded at least once in the test.

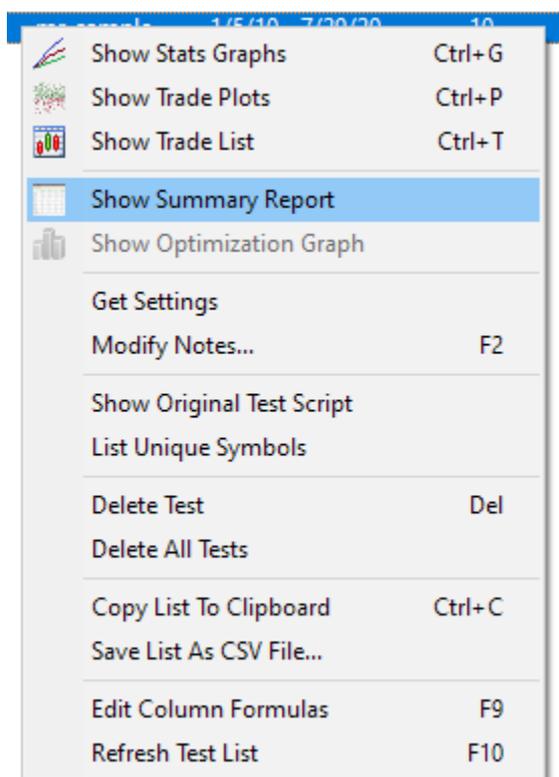
The remaining items control the user interface of the Results window.

## 14.1. Test Summary Report

The Summary Report provides an overview of the results of a test in a unified output format.

There are three ways to create a Summary Report:

1. Run a test with  specified, either via the **Settings Panel** or the **Settings** script section.
2. Select a test in the **Results Window** and then Click  in the **Tool Bar**.
3. Right-click on a test row in a Results Window and select *Show Summary Report* from the menu:



The Summary Report is generated as a web page (HTML and PNG files) and placed in the *Reports* sub-folder of your RealTest installation directory. It is then opened in a new window of your default browser.

The following items are available for inclusion in the Summary Report:

- the settings and parameters with which the test was run
- overall stats for the test (as also shown in its row in the Results Window)
- this same set of stats for each specific strategy within the test
- monthly P/L for each month of the test, organized in a table with one row per year
- strategy correlation matrix, if there were multiple strategies
- a subset of **Daily Stats Graphs** from the test
- a subset of **Trade Analysis Plots** from the test, including Monte Carlo statistics
- the text of the script (plus all included scripts) that was used to run the test

Any of the above categories can be optionally omitted by using the **Report Options Dialog** to deselect its corresponding checkbox before generating the report.

Because these reports are fairly large, an example has not been included in this User Guide.

See **Analyzing Test Results** for pointers to some more detailed and powerful ways to study your backtests.

---

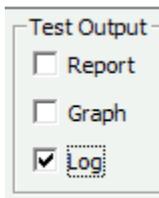
A few Summary Report usage details to keep in mind:

- The selection of Results stats that is included in each row of the Report List index is governed by the presence of `{|}` the stats formula as defined in **Results.rts** or your replacement **Results Section**. These are the same stats that are displayed in the **Status Bar** when the test is running.
- The selection of Daily Stats Graphs to include in Summary Reports is controlled by the item formatting codes in the **Graphs Section**. A graph is included only if its formatting code includes the '^' character. The default **Graphs.rts** script selects the *Equity*, *Drawdown*, *Daily* and *Quarterly* graphs. You can easily change this selection by editing that file and adding/removing '^'s as desired.
- The *Equity* and *Monte Carlo Profit* graphs are automatically plotted using a logarithmic scale if the test was run with compounding, or an arithmetic scale if not.
- All of the *Trade Plots* and *Monte Carlo* stats show percent profits, losses, drawdowns, etc. if the test was run with compounding, or dollar figures of these items if not.
- The generated HTML that displays these graphs and plots includes the ability to click on any image to see the full-sized version of that image.

## 14.2. Test Details Log

---

The *Test Details Log* can be used to study test results at the individual trade level in greater detail than is available in the **Trade List Window**.



To create this log, run a test with `Log` specified, either via the **Settings Panel** or the **Settings**

script section. There is not a way to create a transaction log after a test has been run, it can only be created while the test is running.

The log is created as a file in the "Logs" subdirectory of the RealTest directory, with the name *TEST\_xxxx.TXT*, where 'xxxx' is the test number. If that file already existed, it is overwritten without confirmation, so if you want to be sure to save a transaction log file it is necessary to rename or move it.

When the test finishes running, the test details log file is automatically opened in a new **Log Window**.

The following is some of the log output for the *sector\_etfs.rts* example script backtest:

DATE	TIME	STRATEGY	INFO	SYMBOL	ACTION	QUANTITY	PRICE	FX	SLIP	COMM
1/3/00	open	sector_etfs	\$0 of \$100,000 (0.00%) invested (0.00% exposed) in 0 positions							
1/3/00	open	benchmark	\$0 of \$100,000 (0.00%) invested (0.00% exposed) in 0 positions							
1/3/00	open	sector_etfs	MaxSetups = n/a; MaxPositions = 5; MaxExposure = n/a; MaxInvested = \$100,000; MaxSameCat = n/a; MaxEntr:							
1/3/00	open	benchmark	MaxSetups = n/a; MaxPositions = n/a; MaxExposure = n/a; MaxInvested = n/a; MaxSameCat = n/a; MaxEntries							
1/3/00	open	sector_etfs	5 Setups: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0)							
1/3/00	open	sector_etfs	5 Entries: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0)							
1/3/00	open	sector_etfs	XLB	buy	749	26.70	1		0.0267	3.75
1/3/00	open	sector_etfs	XLF	buy	843	23.74	1		0.0237	4.21
1/3/00	open	sector_etfs	XLK	buy	359	55.68	1		0.0556	1.79
1/3/00	open	sector_etfs	XLV	buy	645	31.03	1		0.0310	3.23
1/3/00	open	sector_etfs	XLY	buy	645	31.03	1		0.0310	3.23
1/3/00	open	benchmark	1 Setup: 1=SPY(0)							
1/3/00	open	benchmark	1 Entry: 1=SPY(0)							
1/3/00	open	benchmark	SPY	buy	674	148.25	1		0	0
1/3/00	intraday	sector_etfs	\$100,031 of \$100,000 (100.03%) invested (100.00% exposed) in 5 positions: XLB; XLF; XLK; XLV; XLY							
1/3/00	intraday	benchmark	\$99,920 of \$100,000 (99.92%) invested (100.00% exposed) in 1 position: SPY							
1/3/00	close	sector_etfs	\$100,031 of \$100,000 (100.03%) invested (100.00% exposed) in 5 positions: XLB; XLF; XLK; XLV; XLY							
1/3/00	close	benchmark	\$99,920 of \$100,000 (99.92%) invested (100.00% exposed) in 1 position: SPY							

Because this log output is tab-delimited, it can optionally be copied to Excel via the clipboard if you find it more convenient to view it in a worksheet and/or want to do further analysis.

To copy an entire detailed log to Excel:

1. Edit / Select All (or press Ctrl+A) in the log window
2. Edit / Copy (or press Ctrl+C) in the log window
3. Edit / Paste (or press Ctrl+V) in Excel

DATE	TIME	STRATEGY	INFO	SYMBOL	ACTION	QUANTITY	PRICE	FX	SLIP	COMM	DIVIDEND	PROFIT	ALLOC	M2M	INVESTED
1/3/2000	open	sector_etfs	\$0 of \$100,000 (0.00%) invested (0.00% exposed) in 0 positions												
1/3/2000	open	benchmark	\$0 of \$100,000 (0.00%) invested (0.00% exposed) in 0 positions												
1/3/2000	open	sector_etfs	MaxSetups = n/a; MaxPositions = 5; MaxExposure = n/a; MaxInvested = \$100,000; MaxSameCat = n/a; MaxEntries = n/a												
1/3/2000	open	benchmark	MaxSetups = n/a; MaxPositions = n/a; MaxExposure = n/a; MaxInvested = n/a; MaxSameCat = n/a; MaxEntries = n/a												
1/3/2000	open	sector_etfs	5 Setups: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0)												
1/3/2000	open	sector_etfs	5 Entries: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0)												
1/3/2000	open	sector_etfs	XLB	buy	749	26.7	1	0.0267	3.75				\$100,000	\$0	\$19,997
1/3/2000	open	sector_etfs	XLF	buy	843	23.74	1	0.0237	4.21				\$100,000	\$0	\$40,012
1/3/2000	open	sector_etfs	XLK	buy	359	55.68	1	0.0556	1.79				\$100,000	\$0	\$60,001
1/3/2000	open	sector_etfs	XLV	buy	645	31.03	1	0.031	3.23				\$100,000	\$0	\$80,016
1/3/2000	open	sector_etfs	XLY	buy	645	31.03	1	0.031	3.23				\$100,000	\$0	\$100,031
1/3/2000	open	benchmark	1 Setup: 1=SPY(0)												
1/3/2000	open	benchmark	1 Entry: 1=SPY(0)												
1/3/2000	open	benchmark	SPY	buy	674	148.25	1	0	0				\$100,000	\$0	\$99,920
1/3/2000	intradav	sector_etfs	\$100.031 of \$100.000 (100.03%) invested (100.00% exposed) in 5 positions: XLB; XLF; XLK; XLV; XLY												

# 15. Trading Your System

---

So you've come up with a set of strategies that work well together and you're ready to start trading them. Now what?

First, please, use the "paper trading" mode of your brokerage interface for a few days (or weeks) to make sure that your system is doing what you think it is doing before committing real money to it.

Even if you plan to fully automate your trading some day, there's a lot to be learned from running a system manually at first.

RealTest does not, at this time, support any type of trading automation.

The following topics show the various ways that RealTest can help with your live trading.

## 15.1. Tomorrow's Orders

---

RealTest includes an **Orders run mode** that you can use to run a script and generate brokerage orders for tomorrow's trading (or today's trading if it's the morning before the market open).

The key point is that RealTest is designed to fully support order generation for *daily* trading systems -- systems for which all orders for the upcoming day can be specified when the market is not currently open. In other words, RealTest does NOT support live trading of strategies that would require live data to evaluate realtime intraday trading signals. In practice, since RealTest can't backtest intraday strategies either, this is not a major limitation to its order generation capabilities.

After a test is run in Orders mode, a new **Log Window** is opened showing the list of orders that would need to be placed with a broker before the open of the next day after the last date of the test. This can also be thought of as *Tomorrow's Orders*.

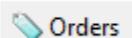
See **Generated Order Types** for details about how RealTest maps strategy rules to generated orders.

To best illustrate this feature, open the **mr\_sample.rts example script**, and modify its *Settings* section to look like the following:

```
▼ Settings:  
DataFile: mr_sample.rtd  
StartDate: Earliest  
StartDate: 2020-01-01  
EndDate: 2020-11-17  
OrdersMode: Text
```

This will model a trader who started trading the strategy at the beginning of 2020 with a \$100K account and needs to know what orders to place for live trading on November 18.

(In actual usage, you would leave the EndDate set to *Latest* and re-import the data file each day.)

After running with these settings by clicking  **Orders**, the *Orders* log window appears:

\*\*\* ORDERS TO PLACE BEFORE THE OPEN OF 11/18/20 \*\*\*

mr\_long exit orders -- change as needed to match actual positions

```
sell 43 ZM market on open (up day)
sell 844 NLS 21.09 limit DAY
sell 309 OTRK 57.53 limit DAY
sell 348 CYRX 51.20 limit DAY
sell 70 IRTC 252.04 limit DAY
sell 243 NARI 73.42 limit DAY
```

mr\_long entry orders

```
buy 246 (10%) [$16,159.74] PLNT 65.69 limit
child: sell 246 PLNT 70.95 limit DAY
buy 643 (10%) [$16,177.88] AQUA 25.16 limit
child: sell 643 AQUA 27.17 limit DAY
```

mr\_short exit orders -- change as needed to match actual positions

```
cover 165 APPN 91.55 limit DAY
cover 403 FLGT market on open (down day)
cover 636 GRWG 23.76 limit DAY
cover 501 EPR 30.22 limit DAY
cover 751 NCLH 20.18 limit DAY
cover 206 SPG 73.55 limit DAY
cover 505 SPR 29.99 limit DAY
```

mr\_short entry orders

```
short 761 (10%) [$-16,186.47] JWN 21.27 limit
child: cover 761 JWN 19.57 limit DAY
short 722 (10%) [$-16,180.02] WKHS 22.41 limit
child: cover 722 WKHS 20.62 limit DAY
short 445 (10%) [$-16,166.85] TUP 36.33 limit
child: cover 445 TUP 33.42 limit DAY
```

Please note that unless you are using **OrderClerk**, this is only a best estimate of what your orders should be. There is no way for RealTest to know what your actual fill prices and quantities were when recent positions were entered, what deposits or withdrawals you've made in your account, and so on.

This pair of strategies use the **MaxSetups** / **SetupScore** technique to ensure that the number of limit orders placed each day does not exceed your available position slots.

These strategies also use *NextOpen* as the **ExitTime** for their rule-based exits (this is the default *ExitTime*).

These two facts are why it is possible to determine "tomorrow's orders" ahead of time.

For strategies that use *ThisClose* entries or exits, it is not possible to foresee what tomorrow's orders will be.

In this case, this caveat will be indicated in the log (here demonstrated by temporarily *ExitTime* to *ThisClose* for these strategies):

\*\*\* ORDERS TO PLACE BEFORE THE OPEN OF 11/18/20 \*\*\*

mr\_long exit orders -- change as needed to match actual positions

```
sell 898 NLS 21.09 limit DAY
sell 329 OTRK 57.53 limit DAY
sell 371 CYRX 51.20 limit DAY
sell 75 IRTC 252.04 limit DAY
sell 259 NARI 73.42 limit DAY
(unable to generate ExitRule orders for tomorrow when ExitTime is ThisClose)
```

Note also the "child: " orders that are added after each entry order in the first order list shown above.

This is because this strategy includes an **ExitLimit** formula (profit target) that might trigger on the same day that a position is entered. In practice, one could place these as "child orders" with a broker, meaning the order is only activated after the parent order has been filled.

If you modify the *ExitLimit* formulas so they don't apply on entry day, the child orders are no longer included:

```
▼ Strategy: mr_long // mean-reversion long strategy
Using: base
Side: Long
EntrySetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMA5) and shares==0// oversold
EntryLimit: LongLimit
ExitLimit: (BarsHeld > 0) * FillPrice * (1 + Target/100) // intraday profit target after first day
ExitRule: C > C[1] or BarsHeld == 5 // cover on first down day or after 5 days
```

no target on entry day

mr\_long entry orders

```
buy 673 (10%) AQUA 25.16 limit
buy 257 (10%) PLNT 65.69 limit
buy 52 (10%) POOL 321.91 limit
```

If a strategy uses **MaxEntries** with unlimited setups, the log text will include a caveat about how many fills to allow:

mr\_short entry orders -- only the first 4 fills

```
short 707 (10%) GRWG 29.16 limit
child: cover 707 GRWG 26.82 limit
short 970 (10%) JWN 21.27 limit
child: cover 970 JWN 19.57 limit
short 921 (10%) WKHS 22.41 limit
child: cover 921 WKHS 20.62 limit
short 568 (10%) TUP 36.33 limit
child: cover 568 TUP 33.42 limit
short 733 (10%) LOVE 28.16 limit
child: cover 733 LOVE 25.90 limit
short 661 (10%) VVI 31.19 limit
child: cover 661 VVI 28.70 limit
short 328 (10%) SLG 62.74 limit
child: cover 328 SLG 57.72 limit
short 538 (10%) PLCE 38.34 limit
child: cover 538 PLCE 35.27 limit
short 970 (10%) SPWR 21.27 limit
child: cover 970 SPWR 19.57 limit
short 899 (10%) NCLH 22.95 limit
child: cover 899 NCLH 21.11 limit
```

(etc.)

### Advanced Live Data Mode

A special advanced mode is also available, which turns this feature into "Today's Orders".

Add **OrdersLiveData: True** to your **Settings** to activate this mode.

Note that doing so assumes that you have found a way to obtain and import today's daily bars for all stocks BEFORE the market has closed, and that your strategies use **ThisClose** as their **EntryTime** and/or **ExitTime**.

## 15.2. Generated Order Types

### Generated Order Types

The following is a complete list of possible order-producing strategy elements showing the specific order types and attributes that will be generated:

Strategy Elements	Related Time Setting	TYPE	TIF	Good Until Time	Good After Time
EntrySetup (no stop or limit)	none or NextOpen	MKT	DAY		
EntrySetup (no stop or limit)	NextClose	MOC or MKT*	DAY		15:58*
EntrySetup + EntryLimit	none	LMT	DAY		
EntrySetup + EntryLimit	NextOpen	LMT	OPG		
EntrySetup + EntryLimit	NextClose	LOC or LMT*	DAY		15:58*
EntrySetup + EntryStop	none	STP	DAY		
EntrySetup + EntryStop	NextOpen	STP	GTD	9:31*	
EntrySetup + EntryStop	NextClose	STP	DAY		15:58
EntrySetup + EntryStop + EntryLimit	none	STP LMT	DAY		
EntrySetup + EntryStop + EntryLimit	NextOpen	STP LMT	GTD	9:31*	
EntrySetup + EntryStop + EntryLimit	NextClose	STP LMT	DAY		15:58
ExitRule	none or NextOpen	MKT	DAY		
ExitRule	NextClose	MOC or MKT*	DAY		15:58*
ExitLimit	none	LMT	DAY		
ExitLimit	NextOpen	MKT	DAY		
ExitLimit	NextClose	LOC or LMT*	DAY		15:58
ExitStop	none	STP	DAY		
ExitStop	NextOpen	MKT	DAY		
ExitStop	NextClose	STP	DAY		15:58

\* order type and time depends on exchange capabilities and settings as defined in *ExchangeMap.csv*

### Special Case for "Day Trade" Strategies

The table above does not cover the ThisClose time setting. That setting is generally not compatible with in-advance order generation as it requires knowledge of the current nearly-completed bar to calculate the strategy formulas.

The one exception to this rule is a type of strategy that enters positions with limit orders and exits that same day with an MOC order as its only exit rule.

To specify this type of strategy so that RealTest automatically generates correct orders, use:

ExitRule: 1  
ExitTime: ThisClose

This will generate a "child" MOC exit order attached to each "parent" entry order.

The entry order is then generated with GTD as its TIF and 15:40 as its expiration time (time can be changed in ExchangeMap.csv).

## Support for International Exchanges, Order Types, and Time Zone Names

RealTest uses an **ExchangeMap** to determine how to correctly map available symbol metadata to correct brokerage orders.

See the above link for full details about the contents and usage of this map file.

The information it contains is used with the **OrdersTemplate** (also with *OrderClerk*) to ensure that:

- symbols are correctly translated from Norgate to IB format
- the correct SMART/xxx exchange definition is used in each order
- MOC orders are generated where supported, otherwise GAT <time> is used
- parent LMT entry orders for MOC exit strategies are placed with GTD <time> as appropriate
- NextClose orders are placed as GAT <time> correctly for each exchange (to become "NearClose" orders)

Note that IB does not require the xxx in SMART/xxx to be the exchange on which a stock is listed. It only needs to pin it to the correct stock market, so that the symbol will not be ambiguous. For example, "SMART/AMEX" works fine for all US stocks even though most of them are not listed on AMEX.

## Norgate to IB Futures Symbol Mapping

RealTest includes a file called **ibfutures.csv**. This file maps **Norgate** futures symbols to the corresponding IB symbols when there are differences.

The default contents are:

Norgate	IB
6A	AUD
6B	GBP
6C	CAD
6E	EUR
6J	JPY
6L	BRE
6M	MXP
6N	NZD
6R	RUR
6S	CHF
6Z	ZAR

If you find others, feel free to add them.

Note that for any Norgate futures contract symbol, RealTest will automatically change the name to the correct IB format when creating a CSV order basket. For example, **ES-2022M** would become **ESM2**, and (also using mapping) **6E-2022M** would become **EURM2**.

To use the **ibfutures.csv** symbol mapping file, you must add **SymChangeList: ibfutures.csv** to your script's **Settings** section.

## 15.3. OrderClerk

---

OrderClerk is an order management application which has been developed specifically to work with RealTest.

A link to the latest version of OrderClerk can always be found in the [RealTest Forum](#).

Currently the only brokerage supported by OrderClerk is Interactive Brokers (IB). Others may be added in the future.

OrderClerk is a smarter substitute for IB Basket Trader, optimized for use with RealTest. It accepts CSV file order lists and transmits them to IB via the TWS/Gateway API.

In addition to placing orders, OrderClerk receives execution reports from IB and maintains a round-trip trade list based on your filled orders. Importantly, each order includes its **Strategy** name, which is then preserved in the trade list.

This enables RealTest to use your live trade list as an input each day when generating tomorrow's orders for a multi-strategy system, ensuring that those new orders are correctly based on the actual current positions and quantities for each strategy.

This capability is implemented by automatically running the script in **Hybrid TradeList Mode** by implicitly adding *TradeList: OrderClerkTrades.csv* (and an appropriate *TLFields* definition) to each Strategy definition.

As an example, to fully automate all seven strategies in the **bensdorp\_book.rts example script**, the following **OrderSettings** section was added to the script:

```
▼ OrderSettings:
  AccountSize: 1e6 // IB paper account default
  StartDate: 5/13/22
  EndDate: Latest
  OrdersMode: OrderClerk
  OrderClerkFolder: Output\Orders\Bensdorp
  OrdersNetLiq: ?ocfolder?\OrderClerkNetLiq.txt
```

The first three items override the general-purpose Settings items used for regular backtesting.

**OrdersMode** tells RealTest to target OrderClerk when generating orders.

**OrderClerkFolder** tells RealTest where to find **OrderClerkTrades.csv** and where to put the new order list.

**OrdersNetLiq** tells RealTest to use the latest actual net liquidation value (mark to market account balance) of the account as the value of **S.Alloc** when generating orders. This ensures that entry order position sizes will accurately reflect the current account value.

The key point is that NO changes were required to any of the strategy definitions themselves. It was NOT necessary to add **TradeList** or **TLFields** statements nor to change any of the regular strategy formulas in order to generate the correct set of new orders. This is all done internally and automatically.

For more information about OrderClerk and how to use it with RealTest, see the separate *OrderClerk User Guide* documentation.

## 15.4. CSV Order Baskets

---

The **Tomorrow's Orders** feature produces an order list in a human-readable text format by default.

RealTest can optionally generate a machine-readable CSV order basket file instead, and lets you define the format using a template file.

This will, in most cases, be an easier and better way to generate a CSV-format order basket vs. building it using a **Scan** or **TestScan**

To create a CSV order basket file, add the following (or something similar) to your **Settings** (note that there is not a way to do this using the Settings Panel):

OrdersMode: **Template**  
OrdersTemplate: **Examples\ib\_basket\_template.csv**

When the test is run, the default text file list of tomorrow's orders will still be displayed, unless you changed that **option**.

The specified CSV file will also be created and formatted using the specified template file.

The default CSV output file will be *{RealTest folder}/Output/Orders/{test\_name}\_{date}\_orders.txt*, where your actual test name and the current date are used.

You can override this default by adding **OrdersFile** to your settings if desired.

When not using **OrderClerk**, an important consideration when using RealTest to generate a transmittable order list is position sizing. Pick a **StartDate** that is as recent as possible while still allowing enough time to align the backtest with your actual positions. In other words, whatever your longest holding period is, you would need to start the test at least that many days ago. You would then also need to set the **AccountSize** setting to whatever your actual account value was on that start date. Even then, the position sizes shown in the order list will most likely not exactly match your live trading. It is always advisable to review and edit as needed the generated files before using them to submit orders.

The *Examples* folder in the RealTest installation directory includes two example CSV basket order template files: **ib\_basket\_template.csv** and **chartist\_api\_template.csv**.

The first can be used to generate CSV order lists that IB Basket Trader can read, and the other will work with the automation "API" that Nick Radge sells as "The Chartist API".

In many cases, using *OrderClerk* will be a simpler and better alternative to either of the above.

The IB Basket template looks like this:

Action	Quantity	Symbol	Exchange	Currency	TimeInFor	GoodTilDc	GoodAfte	OrderType	LmtPrice	AuxPrice	OcaGroup	Orderid	ParentOrd	BasketTag	Account
act	qty	sym	exch	curr	tif	gtd	gat	type	lmt	stp	oca	id	parent	strat	DU1234567

A CSV order list template file should have two rows. The first row will be used as the first (header) row of the output CSV file. The second row defines the fields to include in each order that is added to the list.

In the above example, all of the lower-case text values are special codes for field contents (see below), and the upper-case values are literal constant strings to include.

Running the **mr\_sample.rts** example script from the start of 2022 through Feb-14, the following order list is produced for the next day:

```

Log - C:\RealTest\Output\Orders\mr_sample_20220215_orders.txt
*** ORDERS TO PLACE BEFORE THE OPEN OF 2/15/22 ***

mr_long exit orders -- change as needed to match actual positions

    sell 36 DDS 261.30 limit DAY

mr_long entry orders

    buy 87 (10%) [$8,670.42] ATKR 99.66 limit
child: sell 87 ATKR 107.63 limit DAY
    buy 126 (10%) [$8,719.20] CF 69.20 limit
child: sell 126 CF 74.74 limit DAY
    buy 279 (10%) [$8,738.28] APA 31.32 limit
child: sell 279 APA 33.83 limit DAY
    buy 151 (10%) [$8,751.96] AIG 57.96 limit
child: sell 151 AIG 62.60 limit DAY

mr_short exit orders -- change as needed to match actual positions

    cover 292 CEIX 27.69 limit DAY
    cover 75 ZEN 108.59 limit DAY
    cover 409 APTS market on open (down day)
    cover 260 CRS market on open (down day)
    cover 401 TGI 20.07 limit DAY

mr_short entry orders

    short 43 (10%) [-$8,680.84] CAR 201.88 limit
child: cover 43 CAR 185.73 limit DAY
    short 266 (10%) [-$8,754.06] EBIX 32.91 limit
child: cover 266 EBIX 30.28 limit DAY
    short 255 (10%) [-$8,746.50] ZWS 34.30 limit
child: cover 255 ZWS 31.56 limit DAY
    short 309 (10%) [-$8,741.61] IIIV 28.29 limit
child: cover 309 IIIV 26.03 limit DAY

```

This same set of orders in the generated CSV order basket:

#	Action	Quantity	Symbol	Exchange	Currency	TimelnForce	GoodTilDate	GoodAfterTime	OrderType	LmtPrice	AuxPrice	OcaGroup	OrderId	ParentOrderId	BasketTag	Account
1	SELL	36	DDS	SMART/AMEX	USD	DAY			LMT	261.30		1			mr_long	DU1234567
2	BUY	87	ATKR	SMART/AMEX	USD	DAY			LMT	99.66			2		mr_long	DU1234567
3	SELL	87	ATKR	SMART/AMEX	USD	DAY			LMT	107.63		2		2	mr_long	DU1234567
4	BUY	126	CF	SMART/AMEX	USD	DAY			LMT	69.20			3		mr_long	DU1234567
5	SELL	126	CF	SMART/AMEX	USD	DAY			LMT	74.74		3		3	mr_long	DU1234567
6	BUY	279	APA	SMART/AMEX	USD	DAY			LMT	31.32			4		mr_long	DU1234567
7	SELL	279	APA	SMART/AMEX	USD	DAY			LMT	33.83		4		4	mr_long	DU1234567
8	BUY	151	AIG	SMART/AMEX	USD	DAY			LMT	57.96			5		mr_long	DU1234567
9	SELL	151	AIG	SMART/AMEX	USD	DAY			LMT	62.60		5		5	mr_long	DU1234567
10	BUY	292	CEIX	SMART/AMEX	USD	DAY			LMT	27.69			6		mr_short	DU1234567
11	BUY	75	ZEN	SMART/AMEX	USD	DAY			LMT	108.59			7		mr_short	DU1234567
12	BUY	409	APTS	SMART/AMEX	USD				MKT				8		mr_short	DU1234567
13	BUY	260	CRS	SMART/AMEX	USD				MKT				9		mr_short	DU1234567
14	BUY	401	TGI	SMART/AMEX	USD	DAY			LMT	20.07			10		mr_short	DU1234567
15	SELL	43	CAR	SMART/AMEX	USD	DAY			LMT	201.88				11	mr_short	DU1234567
16	BUY	43	CAR	SMART/AMEX	USD	DAY			LMT	185.73		11		11	mr_short	DU1234567
17	SELL	266	EBIX	SMART/AMEX	USD	DAY			LMT	32.91				12	mr_short	DU1234567
18	BUY	266	EBIX	SMART/AMEX	USD	DAY			LMT	30.28		12		12	mr_short	DU1234567
19	SELL	255	ZWS	SMART/AMEX	USD	DAY			LMT	34.30				13	mr_short	DU1234567
20	BUY	255	ZWS	SMART/AMEX	USD	DAY			LMT	31.56		13		13	mr_short	DU1234567
21	SELL	309	IIIV	SMART/AMEX	USD	DAY			LMT	28.29				14	mr_short	DU1234567
22	BUY	309	IIIV	SMART/AMEX	USD	DAY			LMT	26.03		14		14	mr_short	DU1234567

Notice how the entry limit orders are each given a unique order ID which the child target orders then use to identify their parents. IB Basket Trader translates these basket-specific ID numbers to unique internal values -- all that it requires is that they be unique and express the correct relationships within the basket file.

Comparing the template to the output should serve to clarify how this works. Strings in the second row of the template that match the ones in the following table are placeholders for specific elements of each order (action, quantity, price, etc.) All other strings in the second row are simply copied to every output row, as in the above example.

Here is the list of template field placeholder strings:

String	Description
--------	-------------

date	order date
act	order action (BUY or SELL)
sym	stock or contract symbol reformatted for IB
expiry	expiration date of a contract
act	account number
rtsym	original symbol as shown in RealTest
curr	stock or contract currency
domi	stock or contract domicile (country)
sector	stock economic sector
indu	stock industry
exch	exchange to route orders to (can be e.g. SMART/NYSE)
type	order type (MKT, MOC, LMT, LOC, STP, STP LMT)
qty	position size (shares)
qty0	position size (shares) for entry, 0 for exit
dlr	position size in dollars
frac	position size as an account fraction
lmt	limit price
stp	stop price
tif	time-in-force (DAY, GTD, OPG)
gat	good-after-time time (e.g. 15:58:00 EST)
gtd	good-until-date time (e.g. 15:45:00 EST)
oca	OCA group (automatically added if both stop and limit exits are used)
id	order id (use when attaching target/stop exits to entry orders)
parent	parent order id (use when attaching target/stop exits to entry orders)
primary	listing exchange of stock or contract
sectype	security type of stock or contract
strat	strategy name (typically for the order reference field)
side	side of new or existing position: will be 1 for long, -1 for short
maxent	MaxEntries value (for The Chartist API "Max Fill" column)
exit	whether order is for an exit (1) vs. an entry (0)
note	OrderNote value (could be used as Account, FAGroup, etc.)
extraN	OrderExtraN value, where N is a number from 1 to 3
nocomma	suppresses the comma in the CSV output of this column
tradeid	required for OrderClerk, not needed otherwise

If you require blank columns, simply add them to the header row but leave them blank in the second row of the template.

## 15.5. IB Rebalance Tool

---

RealTest supports generation of the import file for IB's **Portfolio Rebalance Tool**. This is a special version of a **CSV Order Basket**.

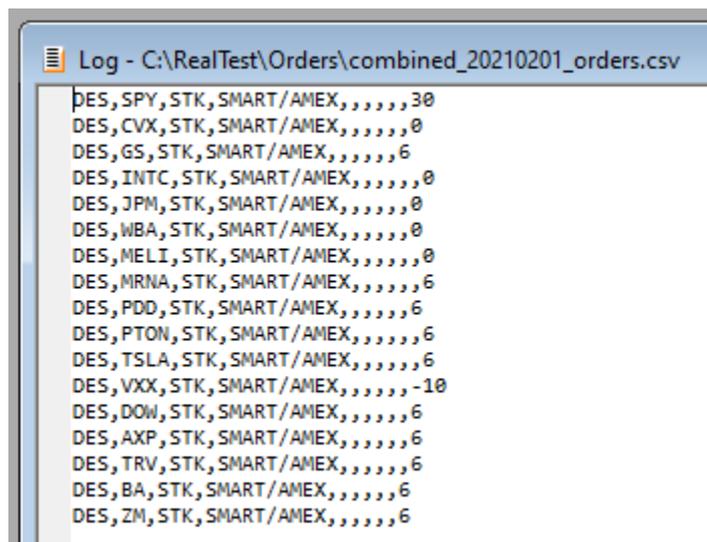
This feature of IB TWS allows you to specify the desired target allocation (percentage) for each symbol in your portfolio.

To use RealTest for this type of portfolio model, all of your strategies must do all of their entries and

exits **At Open**. There is no support for entry or exit limits or stops.

To generate a portfolio rebalance import file, simply add **OrdersMode: Rebalance** to your script's **Settings** section.

Doing so in the **combined.rts** example script and setting its `EndDate` to 2021-01-31 (the end of both a week and month) produces this output when the script is run in **Orders** mode:



```
Log - C:\RealTest\Orders\combined_20210201_orders.csv
DES,SPY,STK,SMART/AMEX,,,,,30
DES,CVX,STK,SMART/AMEX,,,,,0
DES,GS,STK,SMART/AMEX,,,,,6
DES,INTC,STK,SMART/AMEX,,,,,0
DES,JPM,STK,SMART/AMEX,,,,,0
DES,WBA,STK,SMART/AMEX,,,,,0
DES,MELI,STK,SMART/AMEX,,,,,0
DES,MRNA,STK,SMART/AMEX,,,,,6
DES,PDD,STK,SMART/AMEX,,,,,6
DES,PTON,STK,SMART/AMEX,,,,,6
DES,TSLA,STK,SMART/AMEX,,,,,6
DES,VXX,STK,SMART/AMEX,,,,,-10
DES,DOW,STK,SMART/AMEX,,,,,6
DES,AXP,STK,SMART/AMEX,,,,,6
DES,TRV,STK,SMART/AMEX,,,,,6
DES,BA,STK,SMART/AMEX,,,,,6
DES,ZM,STK,SMART/AMEX,,,,,6
```

Adding `OrdersFile: c:\jts\rebalance.csv` to your Settings will make it especially convenient to import this file to IB's tool.

## 15.6. Alera Signal Files

---

The *Tomorrow's Orders* feature produces an order list in either a human-readable **text format** or machine-readable template-defined **CSV format**.

RealTest makes it possible to additionally produce the list of orders for one or more strategies in **Alera Portfolio Manager** signal file format.

To create an Alera signal file for a strategy, add the following (or something similar) to your strategy definition:

```
OrdersFile: c:\Users\Marsten\AleraWorkspace\mhptest\1002\mr_short.sig.txt
```

Note that this is different from the `OrdersFile` specification in **Settings**. Separate strategy-level orders files are supported only because Alera requires them.

Add **OrdersMode: Alera** to your Settings, and run the script in Orders mode to generate your Alera signal files.

Here is an example of a regular text order list and the corresponding Alera signal file:

```
>>> Creating Alera signals file c:\Users\Marsten\Aleraworkspace\mhptest\1002\mr_short.sig.txt for strategy mr_short <<<
mr_short exit orders -- change as needed to match actual positions

cover 42 MATX 66.60 limit
cover 55 SGH market on open (ExitRule)
cover 12 SQ 231.15 limit
cover 43 TWTR market on open (ExitRule)
cover 37 PAR 75.18 limit

mr_short entry orders

short 17 SI 171.19 limit
cover 17 SI 157.50 limit
short 95 APR 32.03 limit
cover 95 APR 29.46 limit
short 135 CONN 22.54 limit
cover 135 CONN 20.74 limit
short 41 LOVE 74.34 limit
cover 41 LOVE 68.39 limit
short 102 HOME 29.75 limit
cover 102 HOME 27.37 limit
short 41 OSTK 74.20 limit
cover 41 OSTK 68.26 limit
```

```
File Edit Format View Help
CLOSE MATX 100 %POSITION LMT 66.60
CLOSE SGH 100 %POSITION MKT
CLOSE SQ 100 %POSITION LMT 231.15
CLOSE TWTR 100 %POSITION MKT
CLOSE PAR 100 %POSITION LMT 75.18
STO SI 17 SHARES LMT 171.19
CLOSE SI 100 %POSITION LMT 157.50
STO APR 95 SHARES LMT 32.03
CLOSE APR 100 %POSITION LMT 29.46
STO CONN 135 SHARES LMT 22.54
CLOSE CONN 100 %POSITION LMT 20.74
STO LOVE 41 SHARES LMT 74.34
CLOSE LOVE 100 %POSITION LMT 68.39
STO HOME 102 SHARES LMT 29.75
CLOSE HOME 100 %POSITION LMT 27.37
STO OSTK 41 SHARES LMT 74.20
CLOSE OSTK 100 %POSITION LMT 68.26
```

Be sure to use the correct Alera signal file path for each strategy in its *OrdersFile* specification.

Once this is set up, all you need to do each day is import a new RTD file with the latest data and then run your script as a backtest.

## 15.7. Daily Setups Scan

While not as simple as just running your system and generating an **Order List**, the use of a daily setups scan has several advantages:

- the output is a **scan window**, which can more easily be saved to a format such as CSV that external automation software can read
- you have total control over the contents of the output

The example script **mr\_sample\_scan.rts** is an adaptation of the **mr\_sample.rts** long/short strategy pair to show how a daily setups scan could be used in live trading:

```

Active Script - C:\REALTEST\Examples\mr_sample_scan.rts*
// example of a daily setups scan for the mr_sample system
// requires Norgate data

▽ Import:
DataSource: Norgate
IncludeList: .Russell 3000 // Current & Past not needed for daily setups
StartDate: 1/2/2019 // only need about a year of data
EndDate: Latest
SaveAs: russell3000scan.rtd

▽ ScanSettings:
DataFile: russell3000scan.rtd
EndDate: Latest
NumDays: 1 // only run scan for the most recent bar for all stocks

▽ Parameters:
NumPos: from 5 to 20 step 5 def 10
PctExt: from 1 to 5 step 0.5 def 2.5
Target: from 3 to 12 def 8

▽ Data:
ATRx: ATR(5)
EMAx: EMA(C,5)
Liquid: C >= 20 and Avg(V, 20) >= 200000
Volatile: ATRx > 0.03 * C
Uptrend: C > Avg(C, Min(BarNum,150))
Biotech: Top(Info(5),4) = 5620 // assumes Norgate data import
// removed constituency test since using current constituent list
Universe: Liquid and Volatile and Uptrend and not(Biotech)
// Long Side
LongSetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMaX)
LongRank: #Rank LongSetup * ATRx / C
IsLong: LongSetup and LongRank <= NumPos
LongLimit: C - 0.5 * ATRx // drops another half ATR
// Short Side
ShortSetup: Universe and C > (1 + PctExt / 100) * Max(O, C[1], EMaX)
ShortRank: #Rank ShortSetup * ATRx / C
IsShort: ShortSetup and ShortRank <= NumPos
ShortLimit: C + 0.5 * ATRx // rises another half ATR

▽ Scan:
Filter: IsLong or IsShort
LongRank: IsLong * LongRank
LongLimit: IsLong * LongLimit
ShortRank: IsShort * ShortRank
ShortLimit: IsShort * ShortLimit

```

The key changes that were made to convert the original example script into this scan script are:

- The **Import Section** has been modified to import only a year of data for the current Russell 3000 components, which is much quicker to use as a daily process than importing the full history with delisted symbols
- A **ScanSettings Section** has been added to specify that this data file be used for the scan and that it is run for only the most recent bar
- The long and short **EntrySetup** formulas have been moved into the **Data Section**
- The **MaxSetups** and **SecupScore** formulas have been converted to data items using the **#Rank** breadth function
- The **EntryLimit** formulas were already calculated in the data section so these are simply used as-is
- A **Scan Section** has been added to select the highest ranked candidates for each strategy and output their ranks and limit prices

After running the import and then the scan, and then sorting the columns by long rank and then short rank, the following is output after the close of 7/10/2020:

Date	Symbol	LongRank	LongLimit	ShortRank	ShortLimit
7/10/20	GAN	1	24.60	0	0
7/10/20	RDFN	2	35.89	0	0
7/10/20	CRNC	3	37.59	0	0
7/10/20	PD	4	27.50	0	0
7/10/20	OSTK	0	0	1	51.31
7/10/20	CAR	0	0	2	29.39
7/10/20	ZYXI	0	0	3	29.83
7/10/20	MATX	0	0	4	38.03
7/10/20	CWH	0	0	5	31.48
7/10/20	BFYT	0	0	6	23.28
7/10/20	TMHC	0	0	7	22.30
7/10/20	RUN	0	0	8	29.88
7/10/20	ONEM	0	0	9	42.98
7/10/20	TSLA	0	0	10	1,595.42

Though up to 10 positions per side are allowed, it appears that only 4 stocks met the LongSetup criteria, while more than 10 met the ShortSetup criteria.

In the original `mr_sample` system, the daily per-strategy cap on the number of new orders to place also accounted for the number of positions currently open. When using the daily setup scan technique, this cannot be done automatically. For example, if you already had 5 short positions open, you would have to either manually remove the last 5 rows from the above output, or temporarily change your short max positions parameter to 5 and re-run the scan.

Similarly, the daily setup scan approach cannot be used to tell you which currently open positions need to be exited. For that purpose, see the next topic: **Test Output Scan**.

## 15.8. Multi-Row Scan

By default, RealTest scans will contain, at most, one row per symbol per date. In certain cases, you might need to output multiple rows for the same symbol on the same date, with different contents. One example of this would be using a scan to generate an order list for IB Basket Trader for a set of strategies that produce more than one order for the same symbol.

To potentially output multiple rows per symbol per date, simply use multiple filter formulas. The maximum number of rows per symbol per date will be the number of different filter formulas provided. As with a single filter, row is only output for any filter if the formula returns a non-zero value ("TRUE") for that symbol on that date.

To add multiple filters to a scan definition, just start the name of each one with the word "filter" and then add one or more other characters to make their names unique. In the example below they're simply called "filter1" and "filter2", but it is not a requirement that they be numbered.

Internally, the scan processor will loop through all the dates in the scan range, then for each date it loops through all the filter formulas (or just does one loop if none is specified), then for each formula it loops through all the stocks. If the stock passes the current filter formula then all the other formulas are evaluated and a new rows is added to the scan output.

When specifying the formulas for the columns that you want to output, you will need to know which filter formula caused the current row to be included. This information is provided by the **FilterNum** syntax element. *FilterNum* simply returns the filter number currently being evaluated. The number returned refers to the sequence in which the filters were listed in the script. Any numbers that happen to be included in the filter names are ignored.

Here's a simplified example of a long/short MR strategy pair using the same set of setups, just different limit prices, configured so each "order" would have its own scan row (some execution engines such as IB basket trader require this):

Active Script - C:\REALTEST\RELEASE\Examples\multi\_filter\_scan.rts

```
// shows how to implement a daily setup scan with (up to) two rows per symbol
```

**ScanSettings:**  
 DataFile: russell3000scan.rtd  
 EndDate: Latest  
 NumDays: 1

**Parameters:**  
 NumPos: 10  
 PctExt: 2.5

**Data:**  
 ATR5: ATR(5)  
 EMA5: EMA(C,5)  
 EMA100: EMA(C,100)  
 Liquid: C >= 20 and Avg(V, 20) >= 200000  
 Volatile: ATR5 > 0.03 \* C  
 Universe: Liquid and Volatile and C > EMA100  
 MyRank: #Rank Universe \* ATR5 / C  
 Setup: Universe and MyRank <= 10  
 LongLimit: C - 0.5 \* ATR5 // drops another half ATR  
 ShortLimit: C + 0.5 \* ATR5 // rises another half ATR

**Scan:**  
 FilterLong: Setup  
 FilterShort: Setup  
 MyRank: MyRank {"Rank"}  
 Action: if(FilterNum == 1, "BUY", "SHORT")  
 LimitPrice: if(FilterNum == 1, LongLimit, ShortLimit)

Date	Symbol	R...	Ac...	LimitPrice
12/11/20	ALXO	1	BUY	71.38
12/11/20	ALXO	1	SHORT	89.42
12/11/20	IGMS	2	BUY	77.44
12/11/20	IGMS	2	SHORT	96.16
12/11/20	ARCT	3	BUY	98.30
12/11/20	ARCT	3	SHORT	113.72
12/11/20	LMND	4	BUY	85.08
12/11/20	LMND	4	SHORT	98.10
12/11/20	BE	5	BUY	23.07
12/11/20	BE	5	SHORT	26.29
12/11/20	EDIT	6	BUY	57.99
12/11/20	EDIT	6	SHORT	65.85
12/11/20	VERI	7	BUY	29.04
12/11/20	VERI	7	SHORT	32.92
12/11/20	NKTX	8	BUY	64.53
12/11/20	NKTX	8	SHORT	72.79
12/11/20	RCKT	9	BUY	56.07
12/11/20	RCKT	9	SHORT	63.07
12/11/20	FMTX	10	BUY	43.47
12/11/20	FMTX	10	SHORT	48.89

Of course for actual basket order generation you'd need extra columns, etc.

## 15.9. Test Output Scan

A regular **Scan** as defined in RealTest only has access to bar data and values calculated in the **Data Section**. The prior topic, **Daily Setups Scan**, showed how a regular scan might be used to generate a list of trading candidates for a strategy.

For situations that require access to strategy context data, such as which positions are currently open, the **TestOutput Scan** option provides additional capabilities.

By default, at the end of a test, RealTest "exits" all positions that remain open. These appear in the **Trade List** with *end of test* as the **Exit Reason**.

When a test is run with the *TestOutput: Scan* enabled and the script includes a **TestScan** section, this special scan is run automatically *before* the *end of test* exits are simulated.

This makes it possible to access all of the same position-context formula elements that are used in your strategy formulas, such as **Shares**, **BarsHeld**, **FillPrice**, etc.

Another use case for TestScan is to generate data throughout a test. For example, you might want to create a CSV file listing each open position on each date of a backtest. To use TestScan in this way, add **TestScanAllDates: True** and **SaveScanAs: <file path>** to your **Settings**.

The **example script mr\_sample\_test\_scan.rts** demonstrates how to use a *TestScan* to produce a daily order list.

See **Tomorrow's Orders** for a much simpler way to do this.

The following shows what was required in before full support for order generation was added to RealTest.

If you were using a script like this for live trading, you would simply run it in *Import* mode and then run it in *Test* mode once per day.

The *TestScan* section in `mr_sample_test_scan.rts` looks like this:

```

▼ TestScan:
Filter: LongEntry or ShortEntry or LongPos or ShortPos
CLS: C {"Close"}
ATR5: ATR5 / C {"%2"ATR"}
Action: if(LongEntry,"BUY",if(ShortEntry,"SHORT",if(LongPos,"SELL",if(ShortPos,"COVER","n/a"))))
Shrs: if(LongPos,LongPos,if(ShortPos,-ShortPos,Round(S.Alloc/NumPos/C,1))) {"Shares"}
Type: if(LongPos and LongExit or ShortPos and ShortExit, "MKT", "LMT")
Price: if(LongEntry, LongLimit, if(ShortEntry, ShortLimit, if(LongPos, if(LongExit, "", LongTarget), if(ShortPos, if(ShortExit, "", ShortTarget), ""))))

```

The purple names are data or library items defined earlier in the script. Here is the **Library** section:

```

▼ Library:
LongEntry: LongSetup and LongRank <= Extern(@mr_long, NumPos - S.Positions)
ShortEntry: ShortSetup and ShortRank <= Extern(@mr_short, NumPos - S.Positions)
LongPos: Extern(@mr_long, Shares)
LongExit: Extern(@mr_long, C > C[1] or BarsHeld == 5)
LongTarget: Extern(@mr_long, FillPrice * (1 + Target/100))
ShortPos: Extern(@mr_short, Shares)
ShortExit: Extern(@mr_short, C < C[1] or BarsHeld == 5)
ShortTarget: Extern(@mr_short, FillPrice * (1 - Target/100))

```

These same items are used in the long and short strategy definitions, which avoids formula repetition:

```

▼ Strategy: mr_long // mean-reversion long strategy
Using: base
Side: Long
EntrySetup: LongEntry
EntryLimit: LongLimit
ExitLimit: LongTarget
ExitRule: LongExit

▼ Strategy: mr_short // mean-reversion short strategy
Using: base
Side: Short
EntrySetup: ShortEntry
EntryLimit: ShortLimit
ExitLimit: ShortTarget
ExitRule: ShortExit

```

Running the script with 17-Nov-2020 as the end date and *TestOutput: Scan* specified in the **Settings** section produces this output:

Date	Symbol	Close	ATR	Action	Shares	Type	Price
11/17/20	APPN	107.74	6.18%	COVER	153	LMT	91.55
11/17/20	AQUA	25.90	5.73%	BUY	385	LMT	25.16
11/17/20	CYRX	46.43	11.47%	BUY	324	LMT	43.77
11/17/20	EPR	31.45	8.40%	COVER	466	LMT	30.22
11/17/20	FLGT	44.43	11.10%	COVER	375	MKT	
11/17/20	GRWG	27.73	10.29%	SHORT	593	LMT	29.16
11/17/20	IRTC	220	8.97%	BUY	65	LMT	210.14
11/17/20	JWN	20.36	8.92%	SHORT	490	LMT	21.27
11/17/20	LOVE	27.01	8.50%	SHORT	370	LMT	28.16
11/17/20	NARI	67.97	8.87%	BUY	226	LMT	64.96
11/17/20	NCLH	22.06	8.05%	COVER	698	LMT	20.18
11/17/20	NLS	18	14.63%	SELL	784	LMT	21.09
11/17/20	OTRK	50.71	8.74%	SELL	287	LMT	57.54
11/17/20	PLCE	36.84	8.12%	SHORT	271	LMT	38.34
11/17/20	PLNT	68.06	6.95%	BUY	147	LMT	65.69
11/17/20	POOL	329.45	4.58%	BUY	30	LMT	321.91
11/17/20	SLG	60.26	8.24%	SHORT	166	LMT	62.74
11/17/20	SPG	78.96	7.80%	COVER	191	LMT	73.55
11/17/20	SPR	33.78	7.99%	COVER	470	LMT	29.99
11/17/20	SPWR	20.44	8.10%	SHORT	488	LMT	21.27
11/17/20	TUP	34.82	8.65%	SHORT	287	LMT	36.33
11/17/20	VVI	29.94	8.36%	SHORT	333	LMT	31.19
11/17/20	WKHS	21.48	8.70%	SHORT	465	LMT	22.41
11/17/20	ZM	401.63	7.89%	SELL	40	MKT	

As you can see from the *TestScan* definition above, the names and contents of these columns can be whatever you want to define them as.

## 16. Backtest Engine Details

---

The RealTest *Backtest Engine* is written to support multi-strategy portfolio-level backtesting as effectively and efficiently as possible.

In most other software, portfolio-based backtesting is implemented by first generating all the entry and exit signals for each symbol for the entire date range, and then running a second pass that models the portfolio for each date using the generated signals.

In contrast to this, RealTest loops through the data by date first, then loops through each strategy, then finally each symbol for that strategy.

Use of this loop hierarchy models a daily trading process in the most realistic way.

Here is a simplified overview of what goes on "under the hood" when you run a test:

1. Apply the **Settings** (if present and applicable) to the **Settings Panel** (where they will persist until next changed)
2. Load the specified **Data File** if not already in memory
3. Determine the current **Parameter** values (the defaults for a single test or the next optimization values for multiple tests)
4. Adjust the test date range to match the data date range and/or the next optimization interval
5. Recalculate **Data Section** items (arrays) as needed
6. Loop through the dates in the test's date range and do the following:
  - a. initialize daily stats
  - b. do at-open **exits** then **entries** for each strategy (exits first to free capital for positions being entered)
  - c. do intraday **entries** then **exits** for each strategy (entries first to avoid assuming capital can be freed when intraday fill sequence is unknown)
  - d. do at-close **exits** then **entries** for each strategy (exits first to free capital for positions being entered)
  - e. update daily stats for each strategy
  - f. recalculate allocation for each strategy

The **exits** in steps b-d above are processed as follows:

1. Loop through all strategy positions and exit each if any of the following are true:
  - a. **ExitRule** evaluates to TRUE and **ExitTime** matches time of day (exit at open or close)
  - b. **ExitLimit** or **ExitStop** price was touched without ambiguity (exit at that price)
  - c. ExitLimit and/or ExitStop price were touched with ambiguity and **Ambiguity** setting permits exit anyway
  - d. this is the last bar of data or last date of the test (exit at close)

In the default "top-down" mode, the **entries** in steps b-d above are processed as follows:

1. For each **strategy**:
  - a. Loop through all stocks and evaluate **EntrySetup**, making a list of stocks for which it returns TRUE (setups)
  - b. Sort this list of setups by **SetupScore** and truncate the list at **MaxSetups** (if specified)

2. For each **setup selection turn number** (loop from 1 to the most setups in any strategy):
  - a. For each **strategy** in **StrategyScore** sequence (highest score first):
    - i. Loop up to **MaxPerTurn** times (default is 1) doing the following steps:
      - 1) select the next unprocessed setup for this strategy (ranked by *SetupScore*)
      - 2) see if adding that setup would exceed any applicable **Max...** constraint for the strategy, any **StatsGroups** it belongs to, or **Combined**
      - 3) if no constraints are exceeded then add this setup and update the strategy, group, and combined constraint values to account for this setup, else skip the setup
3. For each **strategy** (now that we have the complete setup list for all strategies with constraints accounted for):
  - a. Sort setups by **EntryScore** if provided (not necessary in this mode except in rare cases)
  - b. Evaluate **QtyFinal** for each setup to optionally adjust the order quantity now that total setup counts are known
  - c. Evaluate **EntrySkip** if provided and skip the entry if true (ditto)
  - d. Enter the position if **EntryLimit** and/or **EntryStop** have been touched or were not specified

#### Key points about top-down mode:

- position sizes for constraint-checking purposes are always based on **OrderPrice**, not on **FillPrice** (which can't be known yet at setup-processing time)
- all constraints are applied to the setup list prior to processing any entries
- setups for strategies that enter with limit or stop orders count towards all constraints whether or not they are filled
- this model assumes that all orders are placed in advance of the market open each day with no realtime order management (e.g. live cancellation when a capacity is reached)

In **legacy mode**, the **entries** in steps b-d above are processed as follows:

For each **Strategy** in script order:

1. Loop through all stocks and evaluate **EntrySetup**, making a list of the stocks for which it returned TRUE (setups)
2. Sort this list of setups by **SetupScore** and truncate the list at **MaxSetups** (if specified)
3. Sort the setup list by **EntryScore** and loop through the sorted list, adding new positions when ALL of the following are true:
  - a. date matches strategy **BarSize** and time of day matches the strategy **EntryTime**
  - b. **EntryLimit** and/or **EntryStop** price have been touched or were not specified
  - c. there's not already a position in this stock for this strategy (unless **MaxSameSym** is greater than one)
  - d. **MaxPositions** or **MaxInvested** or **MaxExposure** or **MaxEntries** has not been reached
  - e. **EntrySkip** does not evaluate to TRUE

#### Key points about legacy mode:

- all constraints other than *MaxSetups* are applied at entry time, not at setup selection time
- constraints are applied based *filled orders* rather than setups

- this model assumes that *execution software* would manage the constraints in live trading

## 16.1. Asset Allocation and Position Sizing

---

The initial **AccountSize** can be defined in the **Settings** section of your script or via the **Settings Panel**.

By default this value becomes the **S.StartEquity** value and the initial value of **S.Equity** for each strategy and for the combined system.

This models combined compounding of multiple strategies in one account.

To model each strategy compounding separately in its own account, add a **StartPercent** to each strategy.

This partial example shows both of the above alternatives:

```

▽Settings:
  AccountSize:  100000           // combined account size(s) (single or multiple accounts)

▽Parameters:
  AcctPct:      25               // allocate 25% to each strategy
  PosPct:       20               // size positions to 10% of strategy allocation

▽Strategy: shared_account
  Quantity:     AcctPct / 100 * PosPct  // each position in single account gets 20% of 25% = 5%
  QtyType:      Percent
  // default Allocation is Combined(S.Equity) when Startpercent is omitted
  // this implements combined daily compounding of all strategies in one account

▽Strategy: separate_account
  StartPercent: AcctPct           // each account gets 25% of combined initial size
  Quantity:     PosPct           // each position in each account gets 20% of that account
  QtyType:      Percent
  // default Allocation is S.Equity (separate compounding) when StartPercent is present
  // this implements separate daily compounding of strategies in their own accounts

```

As a backtest runs and trades are entered and exited, all of the statistics series including *S.Equity* are updated.

The daily *S.Equity* (equity curve) value for a strategy will be the prior day's value plus the net change in value due to closed trade gains or losses plus (optionally) open position mark-to-market value changes. The combined system equity value is calculated the same way by using all positions from all strategies.

By default, *S.Equity* includes open position mark-to-market value, making it the Net Liquidation Value (NLV) of the account. To model a cash account, in which open position gains cannot be reinvested until the position has been closed, use the strategy-level **MarkToMarket** setting. The default is *MarkToMarket: True*. For any strategy that specifies *MarkToMarket: False*, *S.Equity* will only change when positions are exited. If no strategy marks to market, then the combined system doesn't either.

Each **Strategy** definition will usually include a **Quantity** formula, and can optionally include an **Allocation** formula.

*Quantity* is evaluated at entry time for every new position, and *Allocation* is evaluated at the start of every date of a backtest.

If *Allocation* is not specified, the default is the combined system *S.Equity* value when *StartPercent* was not specified, or the individual strategy *S.Equity* value when *StartPercent* was specified.

In formula terms, this is expressed as *Allocation: Combined(S.Equity)* for combined compounding or *Allocation: S.Equity* for individual compounding.

If the default is your desired allocation, you can simply omit it.

Another *Allocation* possibility is *Allocation: S.StartEquity*. This models an account that trades the same dollar value each day regardless of the results, that is, without compounding.

Default allocation can be overridden if desired by adding an explicit **Compounded** setting to the

strategy.

The current value of the *Allocation* formula is accessible via the **S.Alloc** syntax.

Whether or not you define a custom allocation formula, it is advisable to use *S.Alloc* as your reference to current account value when specifying *Quantity* (position size).

If *Quantity* is not specified, the default position size will be *S.Alloc*, i.e., the entire current allocation.

By default, the *Quantity* formula specifies the number of shares (or contracts for futures) to buy or sell short when entering a new position. The formula can optionally be redefined as either the dollar value of the position or a percent of allocation, by adding **QtyType** to the strategy.

## 16.2. Capacity Constraints

---

RealTest provides several **Strategy Element** formulas to allow you to define the capacity constraints of a strategy or group of strategies in various ways.

These include:

- **MaxPositions** - maximum open position count
- **MaxExposure** - maximum percent exposure
- **MaxInvested** - maximum dollars invested
- **MaxNewPos** - maximum new position count
- **MaxNewExp** - maximum new percent exposure
- **MaxNewInv** - maximum new dollars invested
- **MaxSameCat** - maximum positions in the same category
- **MaxSameSym** - maximum positions in the same security (symbol)
- **MaxSetups** - maximum entry setups per day (processed before other constraints, not top-down)
- **MaxEntries** - maximum position entries per day (potential selection bias, not top-down)

By default, all of the above constraints are **infinite**. Therefore if you want any constraints in your strategy or set of strategies, you must provide one or more of the above formulas.

In the default top-down setup selection mode, most of the above formulas may be included in the definition of a **Strategy**, a **StatsGroup**, or the **Combined** system. In this mode, all constraints are applied during the *Setup Selection* phase of daily entry processing, such that the set of orders to generate each day would violate none of the constraints.

In the older **Legacy Mode**, capacity constraints can only be applied to a *Strategy* and are applied during *Entry Simulation*, modeling an order management system that handles all such constraints. With capacity constraints only applied at the *Strategy* level, each strategy must specify *from the bottom up* how to play its part in modeling your desired higher-level allocation capacities.

See **BackTest Engine Details** for more information about how each of these modes works.

To apply top-down constraints to all strategies in a script, simply add a *Combined* section and define the constraint formulas within that section.

To apply constraints to a specific group of strategies (e.g. all long-side ones), define a *StatsGroup* consisting of those strategies and provide the constraint formulas within it.

In the daily top-down setup selection process, a setup only becomes an order if it does not violate any constraints of its *Strategy*, *StatsGroup(s)* or *Combined*.

The key to how this selection process works is that first all setups are determined for all strategies and ranked by each strategy's **SetupScore** formula. Then an outer loop by selection turn number selects one (or optionally more) setups from each strategy and checks to see if it can still become an order given the others that have already been selected.

If you only provide *Combined* capacity formulas, each strategy will keep the maximum possible number of setups that would (after selecting them in turn) not violate any combined capacities.

Strategy-specific capacities can also be used to further control strategy-level exposures.

Rather than depending on the order of strategies in a script to determine which one "goes first" in the setup selection loop, a **StrategyScore** formula can optionally be provided. When present this formula is evaluated at each rank number in the top-down loop and determines the sequence in which each strategies will get to confirm its next setup.

To enable a strategy to select more than one setup per turn in the selection process, use **MaxPerTurn** to specify how many setups it can select.

One analogy to this *top-down mode* of capacity constraint application is that it resembles the US football annual "draft". Your capacity formulas at each level along with your *StrategyScore* and *SetupScore* formulas provide the framework for deciding who gets the next pick at each stage.

## 16.3. Compounding

---

RealTest automatically detects whether each strategy in a script uses compounding by looking at its Allocation and Quantity formulas.

If a strategy refers to **S.Equity** in its **Quantity** formula, or refers to *S.Equity* in its **Allocation** formula and **S.Alloc** in its *Quantity* formula, then it is considered to *use compounding*.

If neither of these is true, then the strategy is *non-compounded*.

Optionally, this heuristic can be overridden by adding a **Compounded** statement to a strategy.

To find out whether a strategy uses compounding, refer to **S.Compounded** in any formula.

If any strategy in a test uses compounding, then **Combined(S.Compounded)** will always be true.

This combined compounding flag is used internally to determine how some of the results statistics are reported.

When *compounded* is *true*:

- rate of return (ROR) is reported as compounded annual return (CAR) of time-weighted equity
- drawdown percentages are reported relative to the maximum previous equity level, i.e.,  $((\text{current\_equity} - \text{max\_equity}) / \text{max\_equity})$

When *compounded* is *false*:

- rate of return (ROR) is reported as average annual return (AAR)
- drawdown percentages are reported relative to the starting equity level, i.e.,  $((\text{current\_equity} - \text{max\_equity}) / \text{start\_equity})$

This automatic distinction makes it convenient to use either compounded or non-compounded equity without the typical stats distortions which are described at length in the following paragraphs.

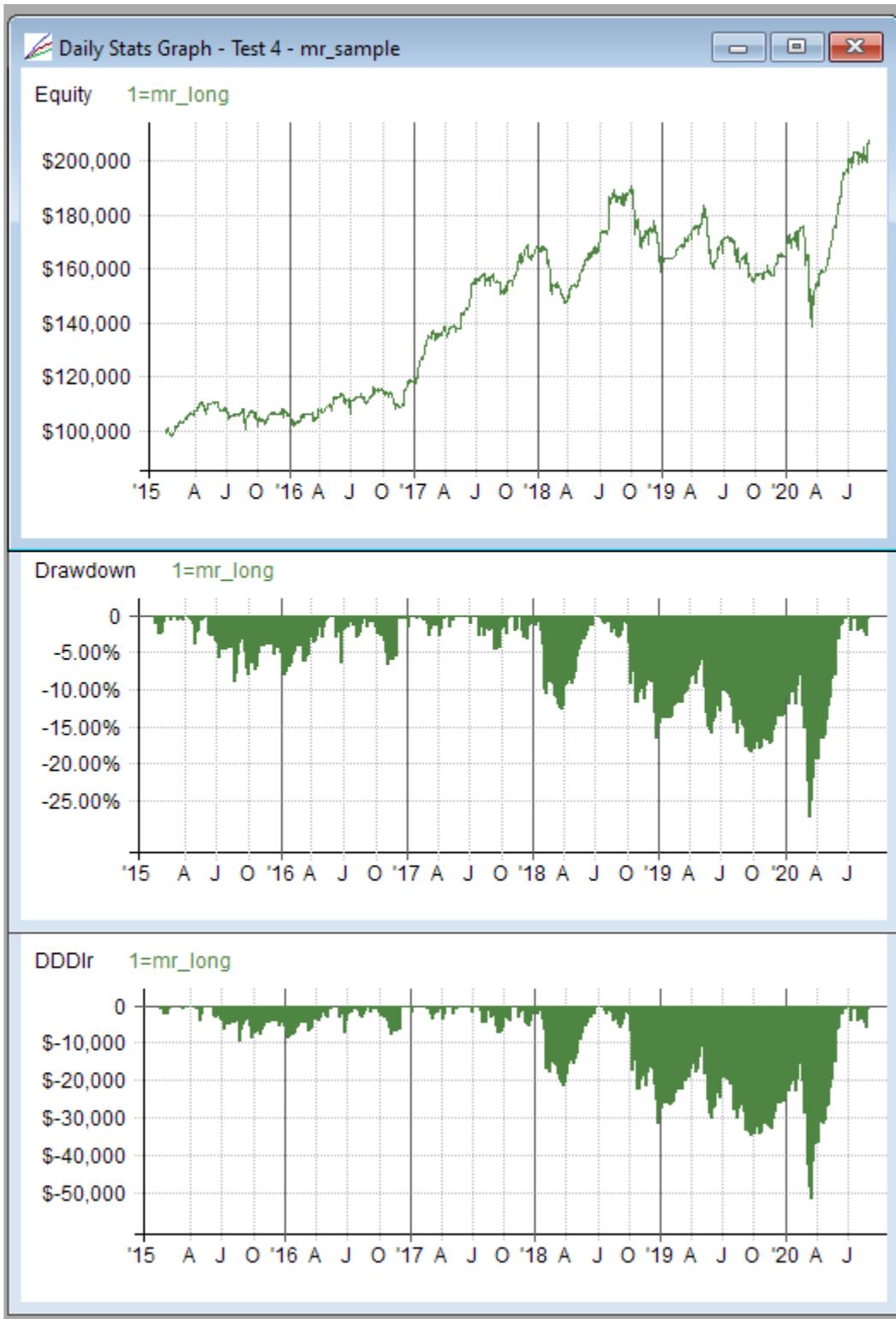
---

Most backtesting software reports "compounded annual return" (CAR) and "maximum drawdown". Both of these stats, as typically reported, assume that a system was modeled to compound its equity (size each new position as a fraction of the current account balance as of the day of entry).

If a system uses a non-compounded sizing approach, such as always trading the same number of shares or the same dollar position value, then these stats designed for a compounding model make no sense. In this case, "average annual return" (AAR) is a better metric, and drawdown percentages should be calculated relative to the starting value of the account rather than its peak (or drawdowns should be shown in dollars rather than percentages).

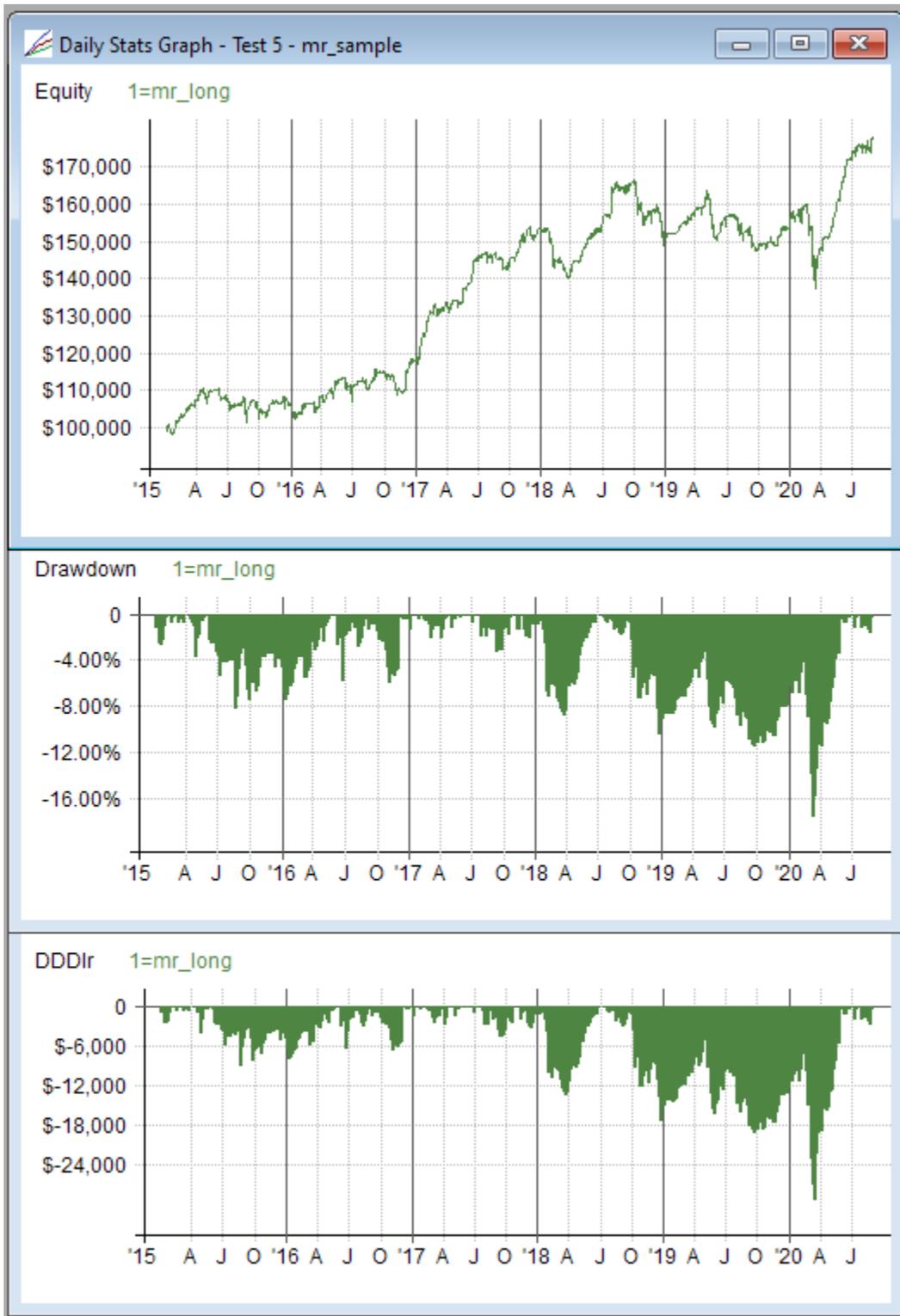
To illustrate, consider the following example.

Here is the **mr\_sample.rts** strategy (long side only) run from the start of 2015 through August 2020:



Because this example uses compounded position sizing, though it started with a \$100K account, the \$50K drawdown in March 2020 was only a little more than 25% from the prior equity peak. As with most compounded backtests, if you compare dollar drawdown with percent drawdown, the ratio of the two gets larger as the date increases (assuming an overall profitable strategy).

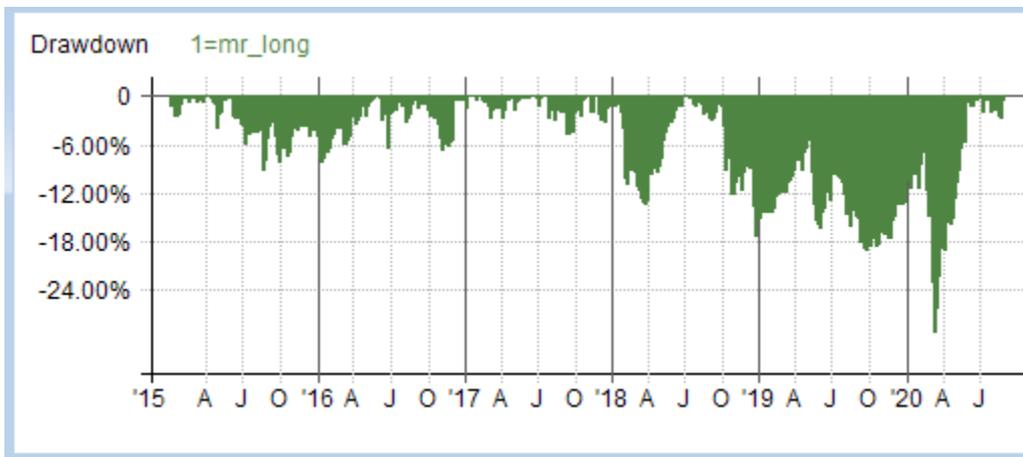
Now if we run this test using non-compounded position sizing (always invest \$10K in each position), the results look like this (after temporarily changing graphs.rts to make it ignore the *compounded* flag and show compounded-style drawdown for this result):



In this case, the \$25K+ drawdown is reported as only a bit more than 16%.

This is the correct percentage based on the equity peak, but it doesn't make sense to report it this way when the model was still using only \$100K of capital at that point in the test.

Changing the drawdown formula back to the correct version, the drawdown percentage graph now looks like this:



In other words, very similar to the same graph from the compounded test.

Looking at the summary results statistics, the default column definitions only include ROR, but here I've added CAR and AAR to illustrate this point:

NetProfit	CAR	AAR	ROR
\$107,877	13.82%	19.08%	13.82%
\$78,163	10.75%	13.82%	13.82%

For the first (compounded) test, the CAR figure makes sense but the AAR value does not (which shows why mutual funds love to report average annual return...)

Conversely, for the second (non-compounded) test, the CAR figure understates the results.

By default, RealTest only includes the ROR column, which automatically displays the appropriate statistic for the compounding mode used in each backtest.

How this is done can be seen by looking at the Results.RTS script:

```

Results:
Periods:      {#} S.Number
NetProfit:    {%0} S.Equity / S.StartEquity - 1
_CAR:        {%2} (S.Equity/S.StartEquity)^(1/(S.Number/252)) - 1
_AAR:        {%2} ((S.Equity / S.StartEquity) - 1) / (S.Number / 252)
ROR:         {%2} iif(S.Compounding, _CAR, _AAR)
MaxDD:       {%2} -S.MaxDDPct

```

(Items that start with underscores are not displayed as columns but can be used to store intermediate values. To produce the output shown above I just temporarily removed the underscores.)

Calculating overall MaxDD percentage is a bit trickier, because it relies (when compounded) on knowing what the peak equity value was prior to the drawdown. This is handled internally, to always return the correct value in the **S.MaxDDPct** stat.

## 16.4. Split Handling

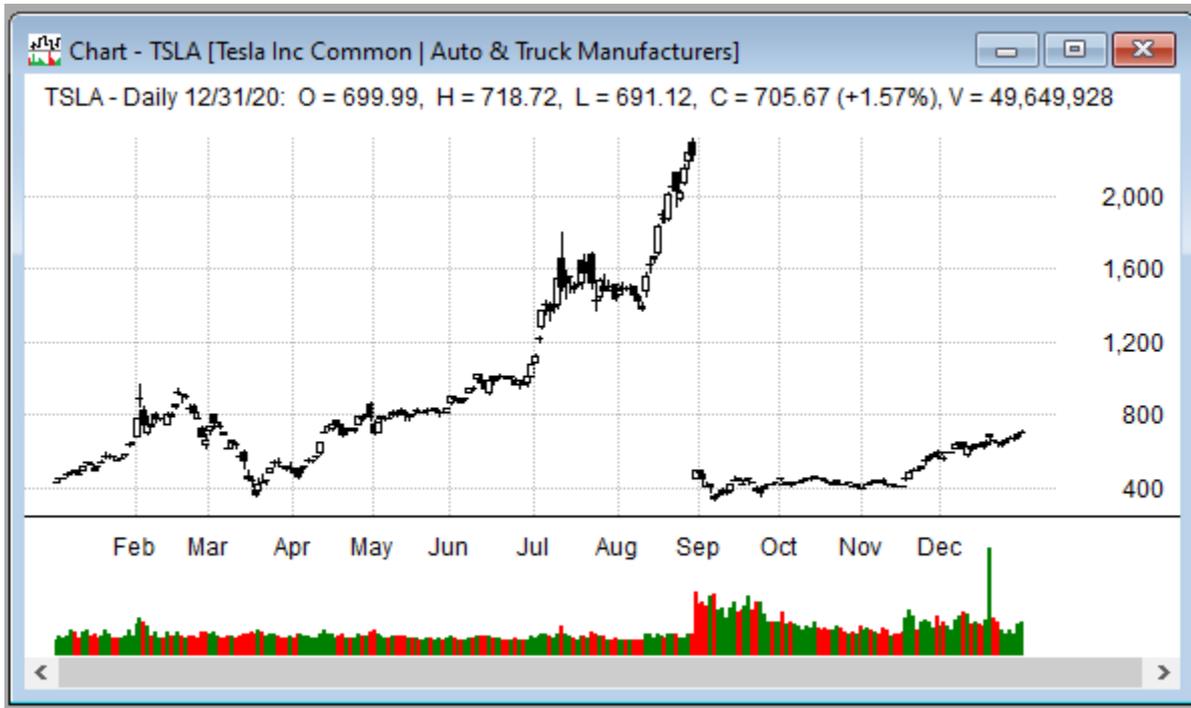
RealTest internally keeps price and volume data split-unadjusted. Any time any formula refers to "Close", the value returned is the real as-traded close for the current bar being evaluated.

To avoid potential distortions when using bar offsets or calculating multi-bar indicators, RealTest temporarily split-adjusts all past bars to the current bar being evaluated while making such calculations. In other words, RealTest always adjusts for past splits, but never adjusts for future splits that could not have been known in advance.

This method of past-only split adjustment makes all price and indicator values automatically "as-traded" in all situations, avoids distortions across past splits, and avoids subtle look-ahead bias which can occur when using data that is adjusted for future splits.

As a simple example, consider TSLA stock in 2020.

There was a 5:1 split on August 31, as shown in this unadjusted chart (charts can optionally be shown either adjusted or unadjusted):



Using the RealTest **Debug Panel**, we can see how any formula would be evaluated for a specific stock on a specific date in a backtest.

Here are two examples of very simple formulas, evaluated first for the day before the split, then the day after:

```
Log - (untitled)*
-----
Symbol:      TSLA
BarSize:     Daily
Date:        8/28/20
Formula:     C
Result:      2,213.40 (2213.4) [price]
-----
Symbol:      TSLA
BarSize:     Daily
Date:        8/31/20
Formula:     C
Result:      498.32 (498.32) [price]
-----
Symbol:      TSLA
BarSize:     Daily
Date:        8/28/20
Formula:     MA(C,5)
Result:      2,128.57 (2128.57) [price]
-----
Symbol:      TSLA
BarSize:     Daily
Date:        8/31/20
Formula:     MA(C,5)
Result:      444.81 (444.81) [price]
```

The first is simply "C", which returns the as-traded close of both dates.

The second is the 5-day moving average, which returns a correct as-traded split-adjusted calculation for each date, thus avoiding the split distortion.

While this all may sound complex, it is complexity that RealTest handles so that you don't have to. Every formula that you use anywhere in a strategy definition simply returns the correct past-only split-adjusted value for each date in the test, as if you had been trading on that day using the latest adjusted data, with no knowledge of future splits.

You may have noticed in the above log output that each result includes both a value (e.g. 441.81) and a type (e.g. "[price]"). In order to correctly handle splits in any formula expression, RealTest needs to keep track of the "type" of each value, i.e., whether it is a price, a volume, or neither. An average of prices needs to be adjusted one way, an average of volume the other way, and an average of price\*volume (approximate turnover) does not require adjustment.

This knowledge of value type during formula evaluation ensures that split-adjustment is correct even when a formula refers to previously calculated data items. Those items know what the final type was when they were calculated, thus enabling their correct adjustment in any other formula.

---

While this mechanism works 99% of the time, it is not perfect. Given the flexibility of RealTest in allowing any formula to be evaluated and have its result stored in the **Data Section**, and then referenced later by another formula, it is possible to find examples where as-needed split adjustment is not correct for previously-stored data items.

One such example is **Extern** symbol values. Using the *TSLA* scenario above, if you were to create a data item called "TSLA\_CLOSE" with *Extern(\$TSLA, C)* as its formula, and then later calculate *MA(TSLA\_CLOSE, 5)* on 8/31/20, the result would not be correct.

Values returned by *Extern* never have a "price" or "volume" type, so storing them in *Data* and then referencing them later with an offset will not provide correct split adjustment.

Another example would be to store a derivative of price that depends on split adjustment in a *Data* item, then refer to it with an offset. For example, during an "exponential slope" calculation such as *Slope(Log(C), 100)*, each of the past 99 values of *C* will be split-adjusted to the current bar before being passed to the *Log()* function. The end result will be correctly split-adjusted for the current bar, as usual with any multi-bar indicator.

However, if you calculate a separate *Data* item like this: *LogC: Log(C)*, and then change the *Slope* formula to *Slope(LogC, 100)*, this will produce a different result if used across a split boundary. This is because *Log(C)*, as a stored data value, can no longer be considered split-adjustable.

The simple rule of thumb to follow, to avoid these rare and obscure potential adjustment errors, is this: do not STORE a price or volume value that can no longer be split-adjusted in a *Data Section* item, and then refer to it using an offset, or as an argument to a multi-bar function or indicator.

---

If you would prefer to "keep it simple" and always work with split-adjusted data, there is a way to do so: just add **KeepAdjusted: True** to your **Import** definition

This will remove the benefits of using unadjusted data, such as realistic as-traded prices and share quantities, and will introduce a risk of look-ahead bias (since adjustments indicate *future* splits), but the option exists if you need it for some reason.

## 16.5. Dividend Handling

---

RealTest can account for dividends in two ways:

### 1. Imported price data can be dividend adjusted

This approach is known as total-return adjustment. Dividends are integrated with the price series by converting them to the equivalent of stock splits.

For example if a \$100 stock paid a \$1/share dividend, that would appear as a 101:100 split. The split price adjustment compensates for the adjustment in the opposite direction that naturally occurs on each ex-dividend day.

Logically this is equivalent to re-investing each dividend payment by buying more shares.

Here is the *sample1.rts* example script output after adding **Adjustment: TotalReturn** to the **Import**, importing, and running:

Active Script - C:\RealTest\SCRIPTS\Examples\Sample1.rts

Notes: Simplest Example -- 50/200 crossover on SPY  
to run for the first time, first click on Import, then click on Test  
to run the test again, just click Test (or press F5)

Import: DataSource: Yahoo  
Adjustment: TotalReturn  
IncludeList: SPY  
StartDate: 1/1/1992  
SaveAs: sample1.rtd

Settings: DataFile: sample1.rtd  
StartDate: Earliest  
EndDate: Latest

Strategy: SPY\_Crossover  
EntrySetup: Avg(C, 50) > Avg(C, 200)  
ExitRule: Avg(C, 50) < Avg(C, 200)

Results - (untitled)

Test	Name	Dates	Periods	NetProfit	Dividends	Comp	ROR	MaxDD	Trades
0001	Sample1	1/29/93 - 9/26/23	7,720	\$1,251,388	\$0	True	8.86%	-33.71%	15

nts	PctGain	Profit	PctMFE	PctMAE	Fraction	Size	SlipIn	SlipOut	Dividends
044	-1.73%	(\$1,734.30)	4.68%	-5.54%	99.98%	\$100,187	\$0.0000	\$0.0000	\$0.00
.54	124.78%	\$123,133.70	170.11%	-5.95%	99.96%	\$98,682	\$0.0000	\$0.0000	\$0.00
.51	24.15%	\$53,318.69	34.12%	-3.65%	99.98%	\$220,777	\$0.0000	\$0.0000	\$0.00
.74	-1.55%	(\$4,233.42)	0.04%	-6.66%	99.97%	\$273,421	\$0.0000	\$0.0000	\$0.00
.59	19.67%	\$52,985.22	25.25%	-3.11%	99.97%	\$269,410	\$0.0000	\$0.0000	\$0.00
.13	16.07%	\$51,935.52	21.23%	-0.26%	99.98%	\$323,115	\$0.0000	\$0.0000	\$0.00
.44	15.62%	\$58,748.56	23.16%	-1.16%	99.97%	\$376,120	\$0.0000	\$0.0000	\$0.00
.47	17.79%	\$77,524.09	33.80%	-6.03%	99.98%	\$435,774	\$0.0000	\$0.0000	\$0.00
.348	0.34%	\$1,712.96	17.56%	-5.03%	100.00%	\$511,178	\$0.0000	\$0.0000	\$0.00
.15	58.22%	\$296,996.21	74.90%	-2.37%	99.98%	\$510,120	\$0.0000	\$0.0000	\$0.00
.33	-6.98%	(\$56,540.72)	1.76%	-9.15%	99.99%	\$810,587	\$0.0000	\$0.0000	\$0.00
.60	33.13%	\$249,735.26	46.46%	-4.97%	100.00%	\$753,911	\$0.0000	\$0.0000	\$0.00
.106	-9.98%	(\$100,153.06)	22.37%	-20.77%	99.97%	\$1,003,282	\$0.0000	\$0.0000	\$0.00
.63	40.47%	\$363,614.31	55.50%	-1.49%	99.98%	\$898,518	\$0.0000	\$0.0000	\$0.00
.95	6.68%	\$84,344.95	14.68%	-5.70%	99.99%	\$1,263,456	\$0.0000	\$0.0000	\$0.00

Notice that there are no dividend payouts.

## 2. Prices can remain as-traded, with dividends imported as events

In the above example in an actual brokerage account, that \$100 stock would be repriced to \$99 on ex-dividend day and you'd be paid \$1/share on the payout day. Assuming no other price movement, this transaction would not change your account value.

To model dividend events in this more realistic way, prices can be imported without dividend adjustment. Dividend amounts are imported as events attached to the bar of each ex-dividend date and containing the \$/share of the payout.

Here is the sample1.rts example script output with the default *Adjustment: Capital* (splits only):

Active Script - C:\RealTest\SCRIPTS\Examples\Sample1.rts

Notes: Simplest Example -- 50/200 crossover on SPY to run for the first time, first click on Import, then click on Test to run the test again, just click Test (or press F5)

Import: DataSource: Yahoo  
 Adjustment: Capital  
 IncludeList: SPY  
 StartDate: 1/1/1992  
 SaveAs: sample1.rtd

Settings: DataFile: sample1.rtd  
 StartDate: Earliest  
 EndDate: Latest

Strategy: SPY\_Crossover  
 EntrySetup: Avg(C, 50) > Avg(C, 200)  
 ExitRule: Avg(C, 50) < Avg(C, 200)

Results - (untitled)

Test	Name	Dates	Periods	NetProfit	Dividends	Comp	ROR	MaxDD	Trades
0001	Sample1	1/29/93 - 9/26/23	7,720	\$1,219,067	\$213,505	True	8.78%	-33.27%	15

ts	PctGain	Profit	PctMFE	PctMAE	Fraction	Size	SlipIn	SlipOut	Dividends
6	-3.17%	(\$3,179.02)	3.97%	-6.73%	99.98%	\$100,187	\$0.0000	\$0.0000	\$1,267.73
13	131.46%	\$127,560.95	152.77%	-5.43%	99.95%	\$97,033	\$0.0000	\$0.0000	\$11,694.05
14	18.05%	\$40,603.22	19.85%	-4.16%	100.00%	\$224,913	\$0.0000	\$0.0000	\$2,821.65
7	2.30%	\$6,136.14	11.85%	-6.53%	99.99%	\$266,385	\$0.0000	\$0.0000	\$2,758.55
12	17.92%	\$48,464.25	23.27%	-3.48%	99.98%	\$270,436	\$0.0000	\$0.0000	\$5,942.25
4	11.35%	\$36,180.17	13.52%	-2.93%	99.96%	\$318,654	\$0.0000	\$0.0000	\$11,282.81
16	17.11%	\$60,924.42	20.65%	-0.15%	100.00%	\$356,168	\$0.0000	\$0.0000	\$11,111.14
17	16.72%	\$70,313.38	35.45%	-3.50%	99.99%	\$420,416	\$0.0000	\$0.0000	\$9,834.27
00	1.58%	\$7,750.78	15.14%	-7.45%	99.98%	\$490,142	\$0.0000	\$0.0000	\$7,545.08
12	59.02%	\$294,093.46	61.60%	-3.89%	99.99%	\$498,336	\$0.0000	\$0.0000	\$46,149.52
5	-4.41%	(\$34,457.50)	2.48%	-6.39%	99.98%	\$780,664	\$0.0000	\$0.0000	\$0.00
13	31.71%	\$239,693.79	40.61%	-4.97%	99.99%	\$755,889	\$0.0000	\$0.0000	\$43,236.51
48	-6.88%	(\$68,455.18)	18.54%	-23.70%	99.97%	\$994,561	\$0.0000	\$0.0000	\$20,138.78
46	36.30%	\$335,892.74	52.71%	-0.73%	99.99%	\$925,329	\$0.0000	\$0.0000	\$25,418.50
19	4.62%	\$57,545.44	11.63%	-7.52%	99.97%	\$1,245,471	\$0.0000	\$0.0000	\$14,303.90

Now we see the dividend payouts. Notice that the NetProfit, ROR, and MaxDD are very nearly the same either way.

There will always be slight differences between the two approaches even for an identical set of trades.

In this case there are also some trade differences. In fact most of the entry and exit dates are different.

Trade date differences are caused by the fact that dividend adjustment influences multi-bar indicator calculations. With this example script, the moving average crossovers happen on different dates in most cases.

Surprisingly, these differences mostly cancel out and the results remain very similar. This will not always be the case. It is up to you to decide which approach you prefer.

To display the total of dividend amounts received, uncomment this line in the default Results.rts script:

```

Active Script - C:\RealTest\RELEASE\Results.rts
Notes: this is the default set of columns for Results windows {...}
Results: // column      {format}      formula
Periods:      {#}              S.Number - S.First + 1 // number of periods
// use S.Number alone if you prefer "Periods"
NetProfit:    {$0}            S.Equity - S.StartEquity - S.CashInOut
// Dividends:  {$0}            Sum(S.Dividends, Periods)
// Interest:   {$0}            Sum(S.Interest, Periods)
// Cash:       {$0}            S.CashInOut // already cumulative
// NetFX:      {$0}            Sum(S.NetFx, Periods)

```

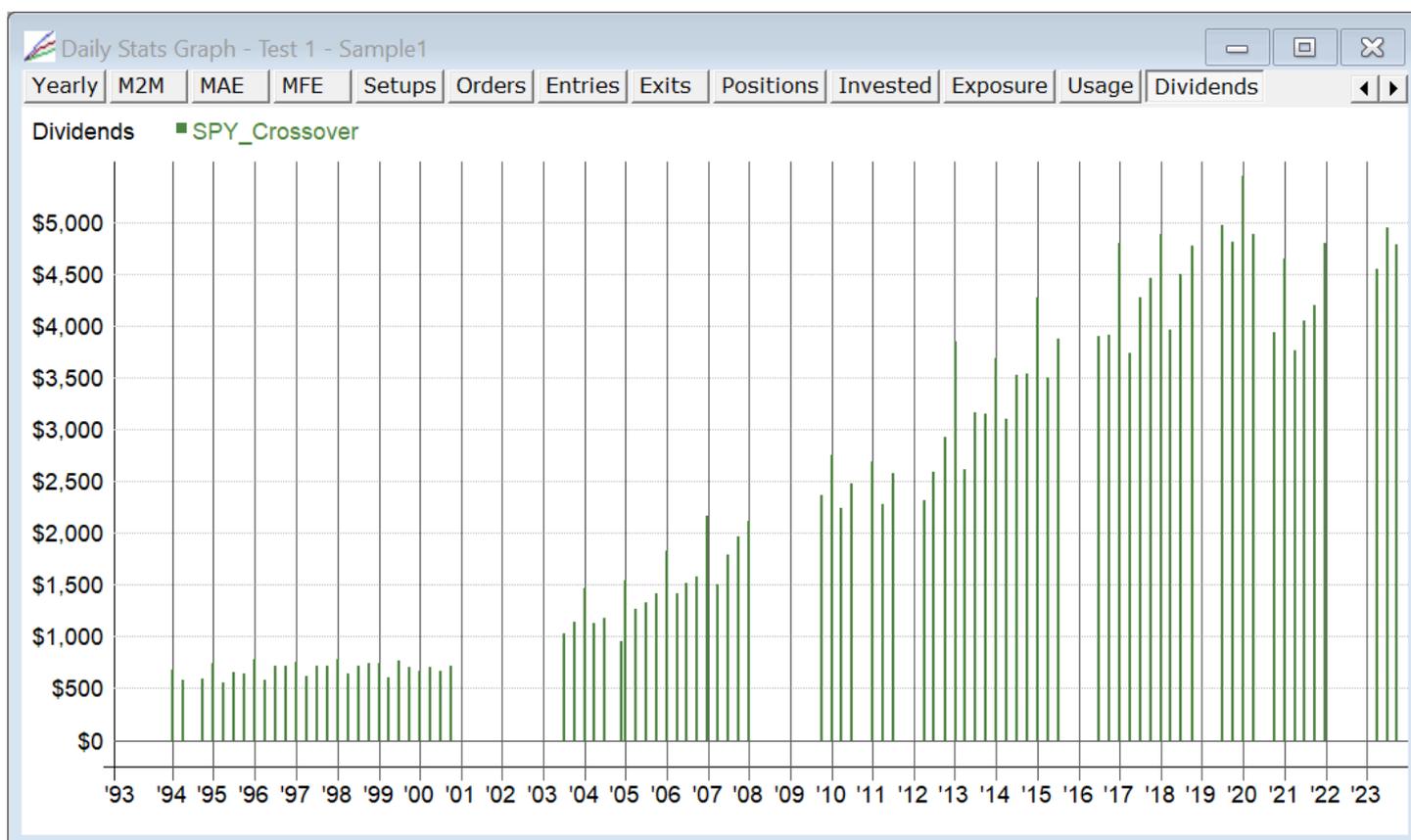
To see daily dividend amounts (combined and by strategy), uncomment this line in the default Graphs.rts script:

```

Active Script - C:\RealTest\RELEASE\Graphs.rts
Notes: this is the default set of stats to offer in Graph windows {...}
Graphs: // title      {format}      formula
Equity:   {$}              S.Equity
TWEQ:    {$^}             S.TWEQ
Drawdown: {%2|^}         -S.DDPct
DDBars:  {#|}             S.DDBars
Daily:    {%|^}           S.NetPct
Weekly:   {%|}            (S.Equity - S.Equity[S.BPW]) / S.Alloc[S.BPW]
Monthly:  {%|^}           (S.Equity - S.Equity[S.BPM]) / S.Alloc[S.BPM]
Quarterly: {%|}           (S.Equity - S.Equity[S.BPQ]) / S.Alloc[S.BPQ]
Yearly:   {%|}            (S.Equity - S.Equity[S.BPY]) / S.Alloc[S.BPY]
M2M:     {%2|}            S.M2M / S.Alloc[1]
MAE:     {%2|}            S.MAE / S.Alloc[1]
MFE:     {%2|}            S.MFE / S.Alloc[1]
Setups:   {#|}            S.Setups
Orders:   {#|}            S.EntryOrders
Entries:  {#|}            S.Entries
Exits:    {#|}            S.Exits
Positions: {#|}           S.Positions
Invested: {$|}            S.Invested
Exposure: {%2|}           S.Exposure
Usage:    {%2|}           S.Usage
// Cash:   {$|}            S.CashInOut - S.CashInOut[1] // cumulative stat
// Dividends: {$|}          S.Dividends
// Interest: {$|}           S.Interest
// NetFx:   {$|}           S.NetFx

```

Apply this change, then click on the Dividends button in your Graph window (you may need to scroll the button bar to see it):



Note that dividends are credited to the account equity as mark-to-market gains on the ex-dividend dates.

An open position that receives a dividend may therefore allow a new position in a different stock to be slightly larger than it would have otherwise been.

These dividend credits are not, however, automatically reinvested in new shares in the open position.

## 16.6. Intraday Fills With Daily Bars

RealTest does not currently support intraday data. It is therefore currently not the best tool if you want to model, for example, a HFT strategy. It is, however, completely practical to develop short-term daily trading strategies and even some intraday strategies (with MOC exit) using daily bars only.

I have spent many years building and running short-term (average holding period 3 days) systems that were tested using daily bars, and have spent a lot of time comparing model to actual results and understanding the differences. The bottom line is that the slight loss of fidelity from not having intraday data in a backtest is dwarfed by the other sources of randomness in live trading.

The standard deviation between model and actual per-trade results is large, but the differences always seem to cancel out, resulting in similar overall results over a large enough sample of trades. My view, therefore, is that most kinds of entry and exit techniques can be modeled using daily bars without compromising the validity of the system (unless, of course, your strategy aims to enter and exit multiple times per day).

If you want to insist on complete fidelity between a model and actual trading, then the only choice available is to enter and exit all positions at the open.

Entries or exits "at the close" are, with today's execution technologies, also completely practical. Orders can be submitted one minute before the close and be filled within seconds, unless you are trading very thin stocks and/or huge size.

Intraday entries and exits (stop or limit orders) can be accurately simulated on daily bars provided that the backtest engine is smart about the following:

1. If the open gaps beyond the price trigger, model the fill at the open (plus slippage), not the trigger price.
2. If the strategy includes both limit or stop entries and limit or stop exits, and the daily bar implies that more than one of the price triggers was hit (within that bar's range), then the model must handle each of the following potential scenarios correctly:
  - a. Limit order entry and stop order exit (loss): it can be assumed that the entry preceded the exit, and both can be filled.
  - b. Limit order entry and limit order exit (target): it can NOT be assumed that the entry preceded the exit, so the target is not hit in the model (unless Ambiguity: Target is specified).
  - c. Stop order entry and limit order exit (target): it CAN be assumed that the entry preceded the exit, so the target is therefore filled.
  - d. Stop order entry and stop order exit (loss): it can NOT be assumed that the entry preceded the exit, so the stop is therefore not hit in the model (unless Ambiguity: Stop is specified).
3. One exception to all cases of (2) is if the stock gaps beyond the entry trigger (1), in which case the entry was at the open so the exit trigger is non-ambiguous.
4. In RealTest's "Default" **Ambiguity** mode, which is not quite as strict as "Neither", it is assumed that:
  - a. if Close > Open, then Low happened before High
  - b. if Close < Open, then High happened before Low

This allows a best-guess assumption to be made in all cases except for a pure doji bar.

The above is how RealTest models all the intraday order types using daily bars. Again, in most cases, none of the above should cause much trouble in strategy development.

## 16.7. Intraday Fill Sequence Assumptions

---

The other factor to consider when modeling intraday entries with daily bars is the order in which fills would have occurred. Say your system identifies 20 or more candidates each day of stocks with a "bull flag" and you want to test buying the first 5 that break above yesterday's high. By the end of the day, 10 of them broke above the high. Which 5 should the model buy? (In practice, running such a system would require either a realtime scan or a mechanism that places the initial 20 orders and then quickly cancels the remaining 15 after the first 5 are filled.)

Some systematic traders worry obsessively about this question and even refuse to trade a system not modeled with perfect fidelity. I would again point out the law of large numbers and the degree of randomness in the market here. If you just look at the next 10 trades, it might matter a lot which 5 of the 10 you assume were filled. But if you look at the next 1000, it makes less difference which 500 are selected.

RealTest of course supports modeling this either way, depending on your preference. The key is to understand what is going on under the hood, be sure it makes sense to you, and be sure your live trading matches the model. Taking some time to study the trade list from each backtest (or even the **Test Details Log** if needed) is the best way to achieve this.

An excellent way to objectively measure the relevance of intraday trade entry order in your system is to use the **Random** function in your **EntryScore** formula, run the same test 100 times, and compare the results.

## 16.8. Bar Sizes and Multiple Timeframes

---

RealTest provides full support for multiple bar sizes within the same data and testing context.

When bar data is **imported**, daily bars are required. Intraday bar sizes will very likely be supported in the future but are not currently. Importing of higher timeframe bars is also not currently support.

However, when daily bar data is imported, weekly and monthly bars are automatically built and any of these three bar sizes is available to use in your scripts.

There is a global bar size setting that is specified either on the **Settings Panel** or in the **Settings** script section. Until changed, this is the default bar size that will be used in all formula elements that reference bar data.

As a simple example, the expression  $C > MA(C,20)$  will compare the daily close to the 20-day average close when BarSize is Daily, the weekly close to the 20-week average close when BarSize is Weekly, or the monthly close to the 20-month average close when BarSize is Monthly.

In addition to the global BarSize setting, there is a strategy-level BarSize setting as well. This is used when you want to combine strategies that use different bar sizes. The strategy-level setting overrides the global setting for all formulas defined within that strategy (including any **Library** formulas they reference, since these are context-dependent, unlike data items).

Regardless of the current global or strategy-level bar size, any part of any expression can explicitly specify a different bar size by using the **Extern** function with the special tilde (~) symbol. For example, *Extern(~Weekly, MA(C,20))* would return a 20-week moving average of close, starting with the most recently completed week and going back 20 weeks from there, regardless of the current bar size setting.

The **Data Section** has both unique capabilities and unique constraints with regard to bar size, which are described [here](#).

For an example of a script that makes full use of multiple bar sizes, see **combined\_multi\_bar\_size.rts** in the **Examples** folder.

---

## 16.9. Calculation of Trade Excursions

---

Trade excursion is the distance from the entry price to the highest high and lowest low that occurred prior to exit. These stats are accessible on a per-trade basis using T.Highest and T.Lowest. Distribution of trade excursions can be studied using the **Trade Plots and Analysis** window.

As with target and stop limit orders, there is potential ambiguity in how these stats are calculated for the specific entry and exit bars.

RealTest uses the following rules for how much of the entry bar to include in T.Highest/T.Lowest if exit is not the same day:

1. Entry at open (whether by design or due to gap beyond limit or stop price): T.Highest starts at High, T.Lowest starts at Low
2. Entry at close: T.Highest and T.Lowest start at Close
3. Entry with long stop, long stop+limit, or short limit (w/o stop): T.Highest starts at High, T.Lowest starts at entry price
4. Entry with short stop, short stop+limit, or long limit (w/o stop): T.Highest starts entry price, T.Lowest starts at Low

For the exit day:

1. Long limit or short stop: if T.Highest was that day, it is changed to exit price
2. Long stop or short limit: if T.Lowest was that day, it is changed to exit price
3. Exit at open: T.Highest or T.Lowest will factor in today's open
4. Exit at close: the day's range is included in both

## 16.10. The Current Bar in Formula Evaluation

---

When writing a formula that refers to bar data elements such as "High" or "Close", it is important understand what "the current bar" is.

In **Data** and **Scan** formulas, the current bar is simply the date for which the formula is being evaluated.

In **Strategy Element** formulas, the current bar is the most recently completed bar relative to the logical time at which the formula is evaluated.

The best way to understand this is to imagine that you're manually trading your strategy by calculating all of your formulas after the close each day. Say today is a Monday. Even though you are preparing to enter or exit positions in Tuesday's market, the "current bar" is still Monday's bar. This fact makes it impossible to accidentally "look ahead" in a backtest to data that you could not yet see in real trading.

An exception to this rule occurs when a strategy uses **EntryTime: ThisClose** (with no entry limit or stop) or **ExitTime: ThisClose**. In these scenarios, RealTest lets you to assume that, by using realtime quotes, you would be able to evaluate your formulas in your live trading platform within a minute or two of the close. In this case and only this case, the current bar is the entry day or exit day bar.

Here is a table of all possible ENTRY scenarios:

<b>EntryTime</b>	<b>EntrySetup Current Bar</b>
ThisClose	entry bar
NextOpen (market default)	bar before entry bar
Intraday (limit/stop default)	bar before entry bar
NextClose	bar before entry bar

All other entry-related formulas use the same current bar as **EntrySetup** does.

Here is a table of all possible EXIT RULE scenarios:

<b>ExitTime</b>	<b>ExitRule Current Bar</b>
ThisClose	exit bar
NextOpen (default)	bar before exit bar
NextClose	bar before exit bar

Here is a table of all possible EXIT LIMIT and EXIT STOP scenarios:

<b>ExitLimitTime or ExitStopTime</b>	<b>Formula Current Bar</b>
ThisClose	exit bar
NextOpen	bar before exit bar
Intraday (default)	bar before exit bar
NextClose	bar before exit bar

**Please take careful note of the following asymmetry between Entry and Exit logic:**

- the Current Bar for each entry-related formula calculation is determined by *EntryTime*
- the Current Bar for each exit-related formula calculation has its own time specification (*ExitTime*, *ExitLimitTime* and *ExitStopTime*)

The reason for this asymmetry is that there is effectively only one *entry order* placed for each *EntrySetup*, whereas there can be up to three *exit orders* placed (with an implied bracket), depending on the presence or absence of each of the three exit-related formulas.

If you intend to use a **Time Stop** in any of your strategies, and/or if you plan to refer to **BarsHeld** in any of your exit formulas, please also read these topics with all of the above in mind.

## 16.11. Specifying a Time Stop

---

The way to specify a **Time Stop** in RealTest is to reference the **BarsHeld** element in the **ExitRule** formula.

For example, a simple five-day time-stop rule would be:

```
ExitRule: BarsHeld = 5
```

If you entered a position on Monday and there are no holidays involved, you will exit the following Monday if you use the above formula.

An easy way to remember this is to think of *BarsHeld* as **Nights Held** (not counting weekend and holiday nights). If you enter on Monday and exit on Tuesday, then *BarsHeld*=1. If you exit on Wednesday, then *BarsHeld*=2, and so on.

When you review the **Trade List** of a test with *Time Stop* exits, you'll see that the **Bars** column always matches the number used in the *BarsHeld* equation for those trades, regardless of their exit time.

Note that this makes *BarsHeld* an **exception** to the logic described in the topic **The Current Bar In Formula Evaluation**. Whereas bar **price** references without an offset always refer to the most recently completed bar, *BarsHeld* includes the current bar in its count even when that bar has not been completed. Specifically, when your **ExitTime** is either *NextOpen* or *NextClose*, *BarsHeld* **includes** tomorrow's bar, even though no prices from that bar are available yet when *ExitRule* is being evaluated.

## 16.12. Number of Bars Required for Functions and Indicators

---

A question that any backtesting framework must answer is: what to do when there are not enough bars of data to evaluate a formula term such as a moving average function. One solution is to place limitations on the syntax, such as requiring that every bar count argument is a constant. In RealTest, there is no need to ever think about how many bars of data are available. What happens automatically is as follows:

Any formula which contains any term which cannot be evaluated with available data simply returns 0. For example, say there's a stock within your data which was only listed 2 weeks ago and you're scanning for  $C > Avg(C, 20)$ . This expression just returns 0 (FALSE), and the scanner moves on to the next symbol.

If you want, you can use **BarNum** to allow shorter average lengths to be used in specific formulas. For example, to scan for "above the 20-day average or the longest average currently available", you could say  $C > AVG(C, Min(20, BarNum))$ . There is also an option, described below, to do this automatically.

It's easy to see how equating "can't evaluate" with 0 works intuitively for conditional formulas like **EntrySetup** or the **Scan** filter. For formulas that return a dollar amount, like **MaxInvested**, or a price, like **ExitStop**, the test engine does the right thing as well. You do not have to worry, for example, that if your long entry stop price formula evaluates to 0, all candidates will trigger an immediate stop. In this and every similar case, the 0 is interpreted as "can never be hit".

All of the above was a long-winded way to say *don't worry about how many bars of data are available*.

An exception to the "number of bars required" rule are the functions **SinceTrue**, **CountTrue**, and **TrueInRow**. These functions treat a *can't be evaluated* term as FALSE but do not abort the entire formula, since doing so would defeat their purpose.

For example, a liquidity filter which I like to use in data import is:

```
ExcludeIf: SinceTrue(C >= 10 and Avg(V, 20) >= 100000) = -1
```

Data import **ExcludeIf** formulas are evaluated once per symbol, after all the bars have been processed, to decide whether to keep or discard that symbol's data.

In this example, stocks will be excluded that have never traded above \$10/share and had average volume of at least 100K shares.

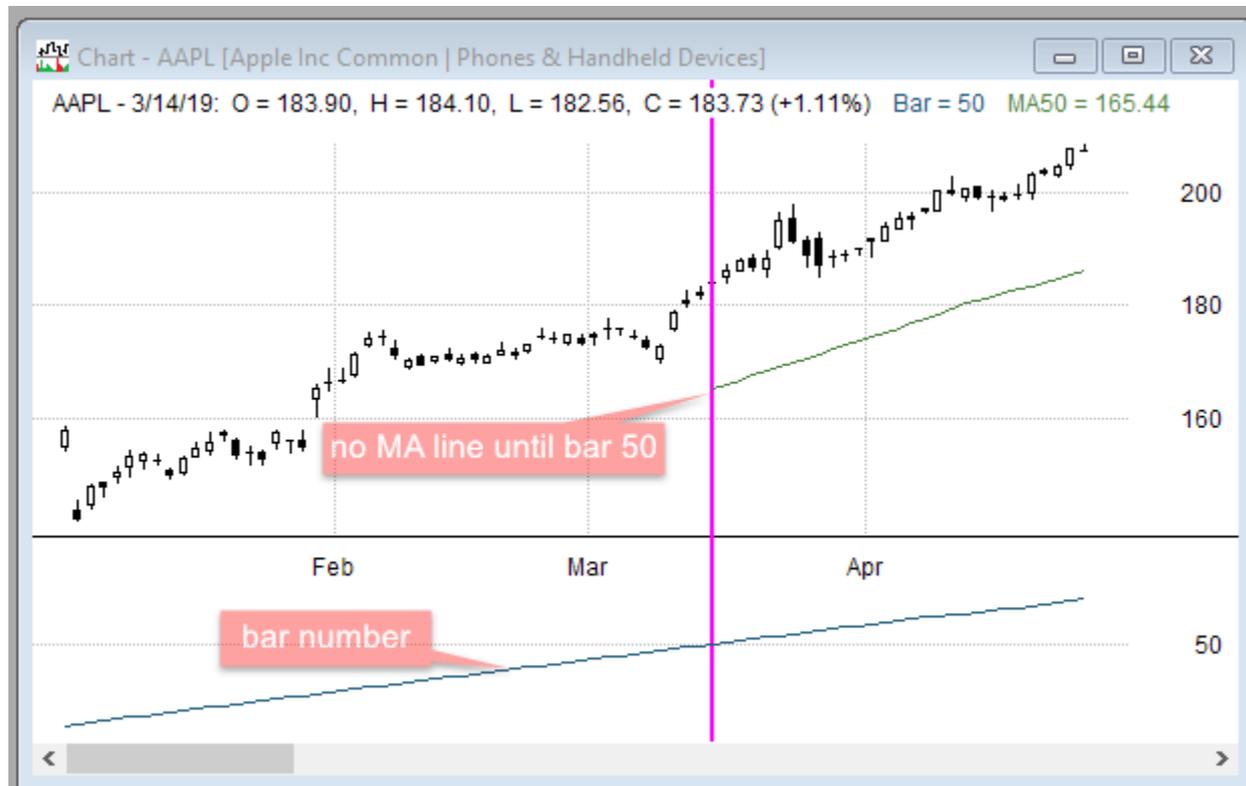
This function returns -1 if a condition was NEVER true within the specified bar count (in this case the default of all available bars).

As mentioned above, there is also an option to automatically shorten moving average and indicator lengths when there are not enough bars available. Most of the time, the default setting (off), as described above, is recommended.

If you know that you want to always use shorter lengths when not enough bars are available (or, in the case of exponential smoothing, permit decreased precision), add **UseAvailableBars: True** to your **Settings** definitions.

You can see this option both ways by plotting a long moving average on a chart and scrolling back near the start of the data. Change the setting and then select *Refresh* from the Chart's menu to see the effect.

With the default setting (off):



With the option to use whatever bars are available:



When using this option, you can also use **BarNum** to explicitly force a specific formula to require enough bars, e.g., `iif(BarNum > 20, Avg(C, 20), NaN)`.

## 16.13. Scaling In or Out of Positions

Most strategies developed and tested in RealTest involve simple round-trip trades. The entire position is bought (or shorted) on entry day, and sold (or covered) on exit day.

RealTest can also model strategies that scale in and/or scale out of positions. Such models can be structured in two different ways:

1. For a small fixed number of entries and/or exits per position, separate strategies can be used for each "leg".
2. A single strategy can enter multiple positions in the same symbol (this "adding to a position") and/or exit part of the total position ("scaling out").

Here's a simple example where we buy at a new 100-day high. The exit is a 2\*ATR trailing stop. We want to sell half the position at a 1\*ATR profit if achieved. Position size is 10% of account.

Here's how to implement this concept with separate strategies:

```

▽Template: common
  Side: Long
  Quantity: 5
  QtyType: Percent
  EntrySetup: C = Highest(C, 100)
  ExitStop: Highest(C - 2 * ATR(5), BarsHeld + 1)

▽Strategy: leg1
  Using: common
  ExitLimit: FillPrice + ATR(5)

▽Strategy: leg2
  Using: common
  
```

The use of a common template avoids the need to copy and paste all of the elements.

The "leg1" strategy holds half the position (5% of account) and implements both exits as a bracket.

The "leg2" strategy holds the other half of the position and only implements the stop.

For scaling in to positions using separate strategies, structure like the above but with different EntrySetup conditions for each leg.

Here's how to implement this concept in a single strategy:

```
Strategy: trail_half
Side: Long
Quantity: 10
QtyType: Percent
EntrySetup: C = Highest(C, 100)
ExitStop: Highest(C - 2 * ATR(5), BarsHeld + 1)
ExitLimit: if(ExitNum = 1, FillPrice + ATR(5), 0)
ExitLimitQty: Shares / 2
```

Here the position size is 10% since there's only one entry. The position will be divided at exit time if the target is hit.

The **ExitNum** variable in the **ExitLimit** formula controls whether a limit price is set (when no exit has happened yet) or not (0 means no limit order).

The **ExitLimitQty** formula specifies the number of shares to sell when the target is hit, i.e., the limit order size.

Note that **Shares** will always represent the share quantity of the remaining position. In the above example *Shares* is only referenced once when the position is still its original size so that's not a problem.

Here's a different scaling-out concept -- sell one-fifth of the original position every day over 5 days:

```
ExitRule: 1
ExitQty: FillQty / 5
```

Using *Shares* in this example would not work as intended. If the original size was 20, the first exit would be 4 leaving 16, the second exit would be 3 (16/5 rounded down), the third would be 2 (13/5), and so on. Instead use **FillQty**, which is the original position size at the time of entry, to specify equal fractions for each leg.

For scaling in to positions in a single strategy, we need to specify **MaxSameSym** as the largest allowable number of simultaneous "legs". By default RealTest sets *MaxSameSym* to 1. This automatically prevents pyramiding without having to add "and Shares = 0" to every **EntrySetup** formula.

This simple entry example would buy the first two times prices crosses above the 8-period EMA:

```
Strategy: cross_ema_twice
Side: Long
Quantity: 5
QtyType: Percent
MaxSameSym: 2
EntrySetup: Cross(C, EMA(C, 8))
```

The **Examples** folder includes a script called **tf\_dynamic\_size.rts**, which demonstrates a mixture of scaling in and scaling out:

```

▼Parameters:
  riskpct:      0.005 // fraction of account to risk in each position
  poschg:      0.05  // quantity fraction change required to trigger resizing

▼Data:
  setup:       InOEX and C = hhv(c,80)
  stopsize:    2 * ATR(5)

▼Library:
  curqty:      extern(@tf_dynamic, shares) // sum of all open quantities for this symbol
  tgtqty:      round(riskpct * S.Equity / stopsize)
  newqty:      extern(@tf_dynamic, tgtqty)
  stop:        HHV(C - stopsize, BarsHeld + 1)

▼Strategy: tf_dynamic
  Side:        Long
  MaxExposure: 100
  MaxSameSym:  99 // multiple sub-positions are required for dynamic sizing
  EntrySetup:  if(curqty = 0, setup, newqty > curqty * (1 + poschg))
  SetupScore:  1/stopsize
  Quantity:    newqty - curqty
  ExitStop:    stop // trailing stop
  ExitRule:    newqty < curqty * (1 - poschg) // reduce size
  ExitQty:     curqty - newqty // shares to remove (may cause multiple sub-positions to be reduced as needed)

▼Benchmark: tf_static
  Side:        Long
  MaxExposure: 100
  EntrySetup:  setup
  SetupScore:  1/stopsize
  Quantity:    tgtqty
  ExitStop:    stop

```

(Not shown are the Import and Settings sections -- the example uses the S&P 100 Current & Past universe.)

This is a far more complex example than the snippets above.

The strategy's **EntrySetup** rule serves two purposes here:

1. Enter a new position if the condition specified in the **Data** section "setup" variable is met and there is no current position.
2. Add a new position portion whenever the target quantity exceeds the current total position quantity by more than X%.

There are two separate exits:

1. The entire position will be exited if the trailing **ExitStop** is hit.
2. A partial exit will occur whenever the total position quantity exceeds the target quantity by more than X%.

As the test runs and partial positions are dynamically added and closed at various sizes, the total position will consist of any number of sub positions. This is why **MaxSameSym** is set to an arbitrarily large number (99).

Notice the special technique in the **Library** section of referring to **extern(@tf\_dynamic, shares)**. This will always return the current total number of shares held by this strategy in this symbol. In contrast, when multiple sub positions are open, "shares" only returns the quantity of the sub position currently being processed for potential exit.

On each day of the backtest, the engine will evaluate the exit formulas for each sub position separately. If a partial or complete exit of that sub position is indicated, that exit is simulated before the engine proceeds to the next sub position. This is why it works to specify the **ExitQty** as is shown above.

The **ExitRule** formula (along with the Library formulas it references) is evaluated for each sub position after the prior sub position's exit has been processed. Once the remaining quantity has reached the target quantity, **ExitRule** will stop returning "true" and no further exits will occur that day.

Note that if **ExitQty** returns more shares than the size of the current sub position, that entire sub position is exited. If **ExitQty** returns fewer shares than the size of the sub position, that sub position is reduced by the difference.

Adding and subtracting sub positions repeatedly in this manner will result, at times in a large number of small sub positions comprising the total current position. Nevertheless the overall result is an accurate model of dynamic position size adjustments.

## 16.14. Referring to Past Trades in Strategy Formulas

The RealTest **Formula Syntax** includes a set of elements which can be used to refer to **Trade Record Values**.

The main purpose of these elements is to facilitate customization of the Trades Window columns by editing the standard **Trades.rts** script or providing an alternative **Trades Section** in a different script.

The trade record syntax can also be used in any **Strategy Element** formula to access certain details of trades which have occurred earlier in the current backtest. This allows you to model strategies which incorporate the concept of looking at past trades in a specific symbol to inform decisions about the next trade.

When used in this way, the context of the trade reference is always the **current bar** of the current symbol in the current strategy. To refer to trades that closed previously, **Multi-Bar Functions** and/or **Bar Offsets** must be used.

If a strategy specifies **MaxSameSym** > 1 to allow multiple positions in the same stock then T.Points, T.Profit, T.QtyIn and T.QtyOut return the sum for all trades with the same exit date (and symbol and strategy).

The following are a few examples of ways in which you might want to explore using this capability:

Example	Description
CountTrue(T.DateOut > 0, 10) > 0	a prior trade in this symbol was exited within the past 10 bars
Sum(T.Profit, 20) < -0.01 * S.Equity	recent trades in this symbol have lost more than 1% of equity
T.Reason = 2	a prior trade in this symbol just hit its target earlier today
Extern(@tracking, Avg(T.Profit > 0, Barnum) < 0.5)	a tracking strategy that takes all signals shows < 50% past win rate for this symbol

For the above examples, the **EntrySkip** formula would be a good place to test these conditions.

## 16.15. Testing Multi-Currency Strategies

RealTest can automatically model buying and selling shares of a stock which trades in a different currency from your account's base currency.

To illustrate how this works, here is a simple example:

```

Active Script - C:\RealTest\Scripts\currency_test.rts
  Import:
    DataSource: Norgate
    IncludeList: MSFT
    IncludeList: AUDUSD
    StartDate: 1/1/21
    EndDate: Latest
    SaveAs: temp.rtd
  TestSettings:
    DataFile: temp.rtd
    StartDate: Earliest
    EndDate: Latest
    Currency: AUD
  Strategy: foreign
    Side: Long
    Quantity: 10000
    QtyType: Value
    EntrySetup: Symbol=$MSFT and BarDate=20210517
    EntryTime: ThisClose
    ExitRule: BarDate = 20210816
    ExitTime: ThisClose
  Charts:
    USDAUD: 1/Extern($AUDUSD, C) {}

```

The script imports MSFT and AUDUSD data series, and specifies the account's base currency as AUD. It then makes one trade, contrived to show a combination of stock and currency gains:

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn	DateOut	TimeOut	QtyOut	PriceOut	Reason
00001	foreign	MSFT	Long	5/17/21	close	31	245.18	8/16/21	close	31	294.60	exit rule

Bars	PctGain	Profit	Fraction	Size	Comm	Div	FxIn	FxOut	NetFx
63	27.46%	\$2,686.83	10.00%	\$9,784	\$0.00	\$22.28	1.2873	1.3631	\$694.01

(the above Trade List row was divided into two images for readability)

Here's the trade on a chart, which also shows USDAUD (1/AUDUSD) in the lower panel:



Looking more closely at this trade, here are all the things that happened:

- The account's base currency is AUD, so the specified position size (value \$10,000) means \$10,000 AUD.
- On entry day, that share quantity is calculated using that day's exchange rate, to buy the appropriate number of shares in USD to not exceed \$10,000 in AUD. In other words, \$10,000 AUD was converted to \$7,768 USD, which could buy 31 shares of MSFT.
- While the position was held, MSFT paid a dividend of \$0.56/share for holders of record as of 18-May-2021. This is credited to the trade profit as  $(31 \text{ shares}) * (0.56 \text{ dividend per share}) * (1.2837 \text{ USDAUD ratio on the record date}) = \$22.28 \text{ AUD}$ , as shown in the "Div" column of the trade list.
- On exit day, the trade's net profit is calculated factoring in the difference in exchange rates between entry day and exit day. In this example, an additional \$694.01 was gained due to the favorable move in USDAUD while the Australian trader held a position in a US stock.

The key mechanics to be aware of in RealTest if you want to do this kind of testing are:

- Your Import must include a symbol for the FX ratio between your account's base currency and the currency of each other country represented in your stock universe.
- It does not matter in which direction the ratio is expressed (AUDUSD vs. USDAUD). RealTest inverts as needed based on the nomenclature.
- If your data source is not Norgate, you might need to use the *alias* mechanism in your **IncludeList** to convert, for example, Yahoo's "JPY=X" symbol to "USDJPY". RealTest requires that the FX symbols are in this six-letter format for this feature to work.
- RealTest similarly requires three-letter currency symbols in the metadata of each stock in order to know whether and how to convert it. Use a SymInfo File if necessary to provide these.
- With Norgate data, the FX symbols are all in the correct format and the currency metadata is provided automatically.
- Add a **Currency** statement to your **Settings**, to tell RealTest that you want it to model currency conversion, and what your account's base currency is. No conversions are done if there is no *Currency* statement.

To observe the impact of currency conversion to your test statistics, the following data are available:

- **T.FxIn** - the exchange rate on the date of trade entry (can be optionally included in the trade list, as in the above example)
- **T.FxOut** - ditto for the trade exit date
- **T.NetFx** - the net dollar impact of FX change to the trade
- **S.NetFx** - the sum of *T.NetFx* for all trades exited that day of the test (can optionally be included in the results summary or graphs)

# 17. Realtest Script Language

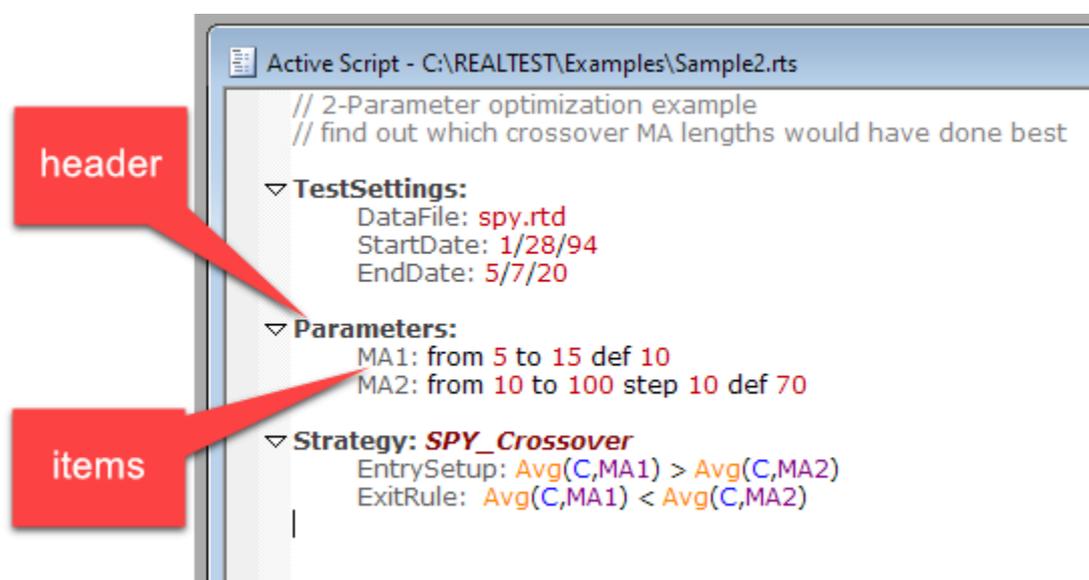
Writing RealTest scripts does not require any programming in a traditional sense.

In fact the script language does not include any standard programming elements such as functions, loops, if/then/else, and so on.

Learning to write RealTest scripts involves primarily the following:

- understanding how scripts are structured (described below)
- learning the **formula expression syntax** (similar to learning Excel cell formulas)
- learning about the **backtest engine** and all of its inputs
- lots of experimentation and reiteration

RealTest script contents are organized into section headers and section items.



```
Active Script - C:\REALTEST\Examples\Sample2.rts
// 2-Parameter optimization example
// find out which crossover MA lengths would have done best
TestSettings:
  DataFile: spy.rtd
  StartDate: 1/28/94
  EndDate: 5/7/20
Parameters:
  MA1: from 5 to 15 def 10
  MA2: from 10 to 100 step 10 def 70
Strategy: SPY_Crossover
  EntrySetup: Avg(C,MA1) > Avg(C,MA2)
  ExitRule: Avg(C,MA1) < Avg(C,MA2)
```

The image shows a screenshot of a script editor window titled "Active Script - C:\REALTEST\Examples\Sample2.rts". The script content is as follows: `// 2-Parameter optimization example`, `// find out which crossover MA lengths would have done best`, `TestSettings:` (with sub-items `DataFile: spy.rtd`, `StartDate: 1/28/94`, `EndDate: 5/7/20`), `Parameters:` (with sub-items `MA1: from 5 to 15 def 10`, `MA2: from 10 to 100 step 10 def 70`), and `Strategy: SPY_Crossover` (with sub-items `EntrySetup: Avg(C,MA1) > Avg(C,MA2)`, `ExitRule: Avg(C,MA1) < Avg(C,MA2)`). A red callout box labeled "header" points to the `TestSettings:` section. Another red callout box labeled "items" points to the `Parameters:` section.

Each section header and each item within a section is declared as a word followed by a **colon**.

Once declared, everything between the colon and the next section declaration becomes the contents of that section.

No statement termination character (such as a semicolon in other languages) is required.

A colon preceded by a word marks both the end of one script element and the beginning of the next.

Although this use of this *Item: content* syntax makes RealTest scripts look slightly like Python, there are no requirements for indentation or line endings. In theory, an entire script could occupy a single line.

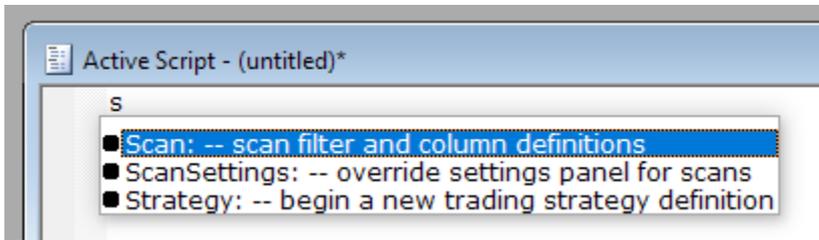
For readability, it is recommended to give each section header and each item its own line, and to indent the items one tab stop. Among other benefits, this permits "folding" in the script editor, and it also facilitates context determination for the context-sensitive help.

One way to think of the RealTest script syntax is as a **form without labels**.

This sounds like a strange idea, but if you imagine a form with all the possible sections and items, it would be overwhelming.

Instead, the intelligent script editor becomes your dynamic form.

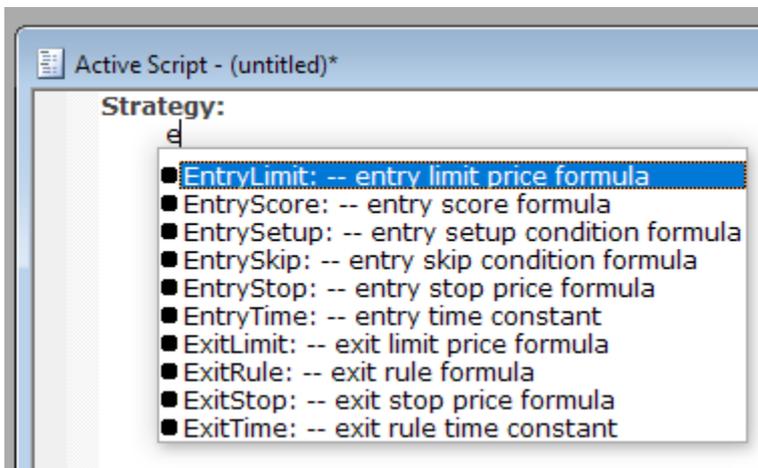
Try opening a new blank script and typing the letter 's':



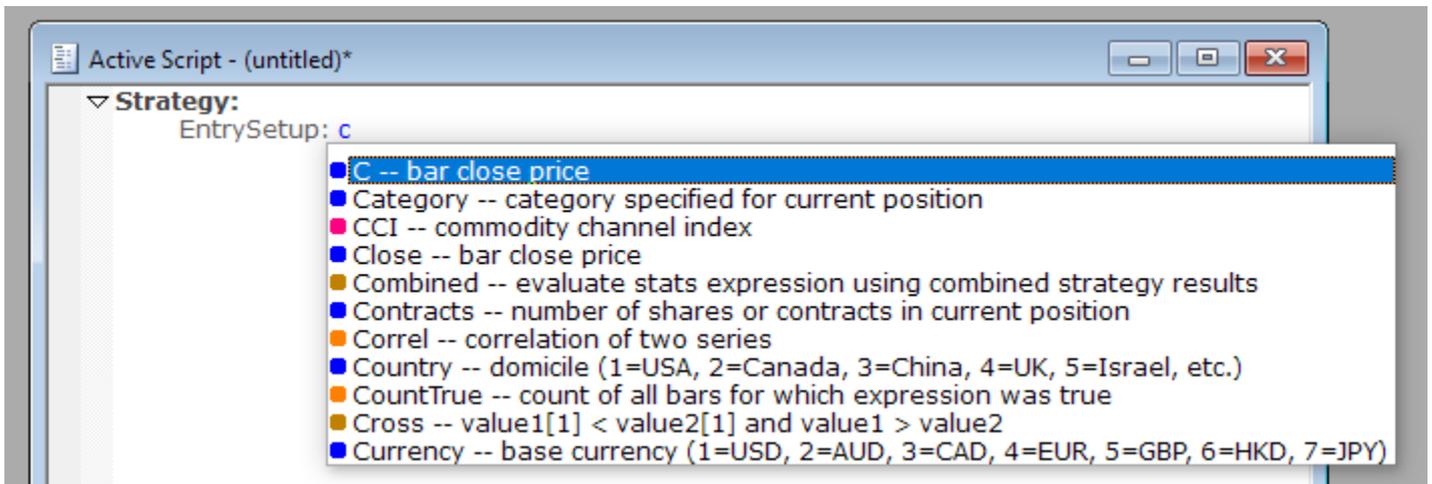
Immediately you have a choice of three possible new *form labels*, aka *section names* to insert.

Select *Strategy*, and then press enter. The cursor moves to the next line and is automatically indented one tab stop.

Now type 'e' and see all the strategy labels that contain that letter:



Select *EntrySetup*, then type 'c' to begin your formula:



I find this interface much more convenient and easy to use than a large form with lots of labels on it. I hope you do to.

## 17.1. Bar Offsets

When a bar field such as *Close* or *Volume* is referenced in a formula, there is always a "current bar" context.

In a test, for example, the **backtest engine** is looping through the dates of the test range and evaluating formulas such as **EntrySetup** for each stock that has a bar for that date.

So when a formula uses a simple term like *Close*, it is inferred to be the closing price of the current stock being evaluated for the current date in the test.

When you need to refer to a different bar of the current stock, you do so using an *offset*.

Some software, including *MetaStock* and *AmiBroker*, use a function called "Ref" when an offset is required, and express the offset as a negative number of bars. Yesterday's close would be  $Ref(C, -1)$ , the prior day is  $Ref(C, -2)$ , and so on.

Other software, including *TradeStation* and *ThinkOrSwim*, use square brackets to express an offset, and express it as a positive number of bars. Yesterday's close is  $C[1]$ , the prior day is  $C[2]$ , and so on.

RealTest uses the square bracket syntax, but with some important enhancements:

- The value inside the brackets can be *any expression*. Say you wanted some random close between 1 and 10 bars ago. That would be  $C[Random(1,10)]$ .
- Brackets can be applied to any expression. The difference between yesterday's high and yesterday's close could be expressed as  $H[1]-C[1]$ , or as  $(H-C)[1]$ . The 10-day average close as of 3 days ago would be  $MA(C,10)[3]$ .
- Though not generally advisable, negative offsets can be used to look into the future. Tomorrow's close is  $C[-1]$ , the day after tomorrow is  $C[-2]$ , and so on.
- One example of a valid use of negative offsets to obtain future values is with date elements such as **BarDate**, **DayOfWeek**, **EndOfMonth**, etc.

## 17.2. Date Constants

---

Date constants can be formatted in any of the following ways:

- mm/dd/yy (or dd/mm/yy if **that setting** is specified)
- mm/dd/yyyy (or dd/mm/yyyy ditto)
- either of the above with dots (.) in place of slashes (/)
- yyyy-mm-dd
- yyyy.mm.dd
- yyyymmdd
- dd-mmm-yy (where mmm is the first three letters of the month name)
- dd-mmm-yyyy (ditto)
- Earliest (use the earliest available date as a StartDate)
- Latest (use the latest available date as an EndDate)

Date constants are used in the **Import** and **Settings** script sections.

When using a date in a formula, e.g.  $if(Date = 20200607)$ , it is simply a number and must therefore be formatted as YYYYMMDD.

If you prefer, you can use **ToDate** to convert date strings in any of the above formats (except Earliest and Latest) to numbers.

Any type of CSV files that RealTest reads can use any of the above formats (other than "Earliest" and "Latest") in their Date column values.

To override the current date format setting (M/D/Y vs. D/M/Y) in a **CSV import**, add a **CSVDateFmt** definition.

To override the current date format setting a strategy's imported **TradeList**, add a **TLDateFmt** definition.

## 17.3. Other Constants

---

Numeric constants can optionally be expressed as a power of 10, e.g. "1E7" rather than "1000000".

The constants *True* and *False* are built in and can be used anywhere as substitutes for 1 and 0 respectively.

The constant *nan* is built in and can be used to force a formula result to be nan (not a common need).

Most **Strategy Elements** are defined by **Formula Syntax**, but many are defined by constants that represent the selected choice for that setting. See **Strategy Element Value Types and Defaults** for details.

## 17.4. Symbol Constants

---

The **Symbol** syntax element returns the alphabetical ordinal number of the current symbol in the currently loaded **DataFile**.

The main use of this number is to compare it to a specific symbol.

For this purpose, every symbol in the current data set can be referenced as a constant beginning with a dollar sign, e.g. **\$MSFT**.

Symbol constants simply return the ordinal number of the specified symbol if it exists in the current data, or 0 if not.

The most common use of symbol constants is when constraining a strategy to only trade a specific symbol, as in this example:

```
▼ Benchmark: SPY // buy-and-hold SPY with dividend reinvestment  
  EntrySetup: Symbol = $SPY // simply enter SPY whenever there is no position  
  ExitRule: Dividend > 0 // exit (and immediately re-enter) on each ex-dividend day
```

Note that if a symbol already begins with a dollar sign, e.g. \$SPX, then it must be stated with a double dollar sign, i.e. **\$\$SPX**, when used as a symbol constant. The first \$ tells RealTest that you're looking for a symbol, and the second \$ is literally part of the symbol.

See also:

- **?Symbol** - returns the current symbol as a string (without the leading \$)
- **SymNum** - looks up a symbol string (without the leading \$) and returns the Symbol number

If the current symbol is MSFT, then any of the following will return its symbol number:

- \$MSFT
- Symbol
- SymNum(Symbol)
- SymNum(?Symbol)
- SymNum("MSFT")

## 17.5. String Values

---

RealTest allows literal quoted strings to be used in situations where they are useful (in particular, for **Scan** output formatting).

However, it does not require (or allow) quotes to be used around strings that are common elements of many scripts.

Script elements that do not require (or allow) quotes around them include:

- data source names
- file paths (even when spaces are embedded)
- stock symbols
- strategy names

Quoted strings can be used as terms in any formula, and can be either 'single-quoted' or "double-quoted". If you need literal quotes of one type within a string, surround it with the other type.

Basic formula operators that work with strings are: <, <=, >, >=, ==, <>. For example, 'b' > 'a' will evaluate as 1 (true). In all cases, case-sensitive comparison is used. (For case-insensitive comparison, use ToLower('B') > ToLower('a'), for example.) You are not prevented from using other operators with strings, comparing strings to numbers, etc. but the results of doing so are unlikely to be meaningful.

Several functions are provided to operate on strings, including:

- Format
- Match
- ToDate
- ToNum
- ToLower
- ToUpper

The most likely use of strings as mentioned above is in **Scan** output. They can also be used in **Trades** column definitions. You can experiment with string expressions in the **Debug Panel**.

## 17.6. Script Comments

---

The RealTest script syntax supports three different comment formats (for no particular reason):

- C style line-independent comments: `/* this is a comment */`
- C++ style single-line comments: `// this is a comment`
- Pascal style curly brace comments: `{this is a comment}`

Each type of comment ignores the other types.

The editor includes the ability comment or un-comment a block of lines by adding or removing a C++ style comment on each line. It is often convenient to use this feature to comment out some of the strategies in a script while focusing on different ones in your research, then later comment them back in.

A special version of the curly brace comment style is also used in some formulas to specify **number formatting** in results, graphs, charts and scans.

## 17.7. File Path Syntax

---

There are several places in a RealTest script where a file path is called for.

Script elements that require a file path include: **DataFile**, **IncludeList**, **TradeList**, **SaveScanAs**, and several others like these.

Unlike most other software where file paths are specified, RealTest neither requires nor allows quotation marks around the path, even when it contains embedded spaces. This saves you from having to remember to type quotes, and makes it more natural for the script editor to suggest multiple choices when you've typed part of the path.

As mentioned under **File Paths** earlier, RealTest supports both absolute and relative file paths.

Absolute paths typically begin with a drive letter, e.g. *C:\RealTest\Scripts\Examples\Combined.rts*

Relative paths might be just a file name, e.g. *Combined.rts*, or the end of a path, e.g. *Examples\Combined.rts*

When relative paths are used, they are relative to your *Script Path*, *Data Path*, or *Output Path* as specified in the **Program Options Dialog**.

RealTest also supports a special syntax within any file path that can be used to insert a known item into the path. The following table lists these elements.

Item	Description
?date?	the current date in YYYYMMDD format
?time?	the current time in HHMMSS format
?orderdate?	the next market date after the end date of a test (for order list files)
?ocfolder?	the full path of the current <b>OrderClerkFolder</b> setting
?scriptpath?	the full path of the current script, without the name
?scriptname?	the full path of the current script, with the name
?testname?	the name of the test just run (for SaveStatsAs etc.)
?desktop?	the full path to your Windows desktop folder

Additionally, any Windows environment variable can be specified, e.g. ?programdata?, ?onedrive?, etc.

## 17.8. Output Format Specification

---

Several of the **Script Sections** in RealTest serve to define items which are both calculated and displayed.

These sections include: **Charts**, **Graphs**, **Results**, **Trades** and **Scan**.

In each of these sections, the format in which to display the value of the calculated formula can be optionally specified by including a special kind of comment within the formula definition.

A format specification comment must use the squiggly-brace **comment style**.

If no format specification is provided, the "general" format will be used (as in Excel).

As one quick example, the *NetProfit* formula in Results.rts is defined as  $\{\$0\} S.Equity - S.StartEquity$ . The \$0 tells RealTest to display this stat as a whole dollar value.

You can see other examples of these codes in the section links above.

### Available Formatting Codes

Code	Used In	Shows This	Digits (if present) Specify
#	scans, trades, charts, graphs, results	general-purpose numeric value	number of decimal places
##	scans, trades, charts, graphs, results	general-purpose number without commas	number of decimal places
\$	scans, trades, charts, graphs, results	value as currency	number of decimal places

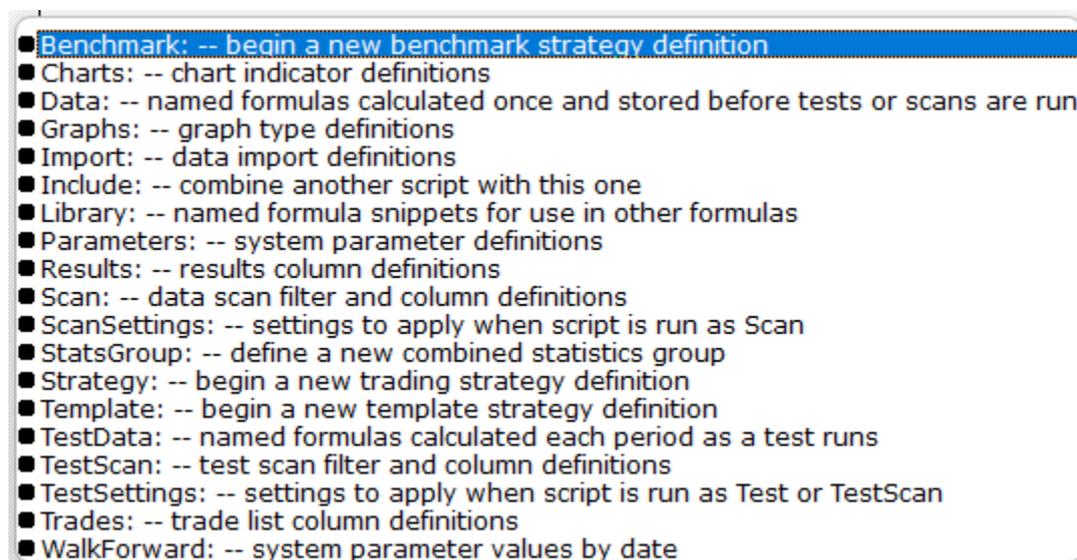
%	scans, trades, charts, graphs, results	value * 100 as a percent	number of decimal places
-	with any of the above codes	value as above, use red if negative	the ABS is the number of decimal places
//	scans, trades, charts, graphs, results	value as a date string	n/a
~	scans, trades, charts, graphs, results	value as "True" or "False"	n/a
^	charts	show this indicator in the upper indicator pane	n/a
	charts	show this indicator in the volume pane	n/a
	graphs	draw bars rather than a line	base value (x-axis) for bar chart
	results	show this item in the status bar	n/a
"name"	any	override the default item name	n/a

An output format specification comment can be inserted anywhere within the item's formula. Any unrecognized character in the comment cancels its interpretation as an output format.

## 17.9. Script Sections

The outer sections of a script represent different categories of script functionality.

To quickly see a list of all available sections, press F2 in a script window with the cursor at the start of a blank line:



The following topics describe each script section in more detail.

### 17.9.1. Import Section

The items within the Import section are used to specify everything about data import.

See **Importing Bar Data** (and sub-sections) for a detailed explanation of how data importing works in RealTest.

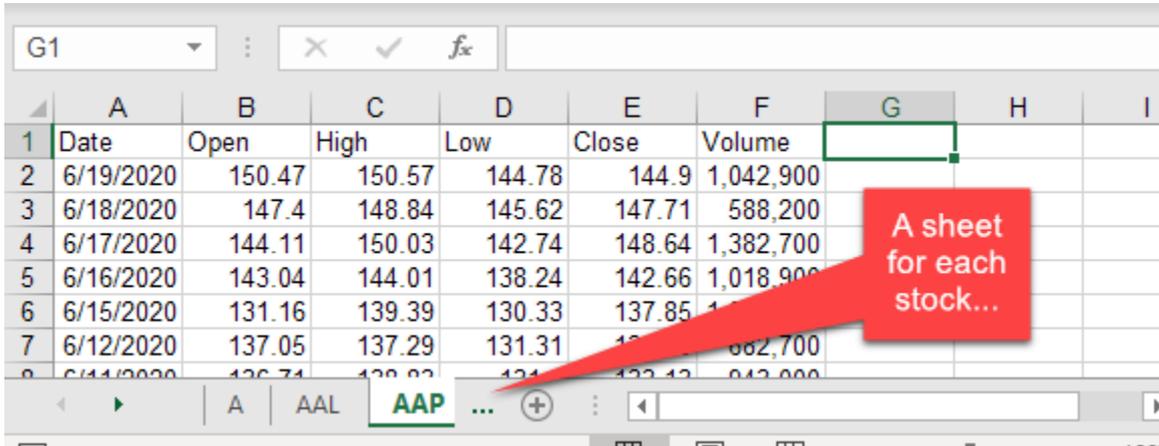
See **Import Specification** for an alphabetical list of Import section elements.

## 17.9.2. Data Section

The Data section is probably the most important feature of RealTest. Fully understanding how this section works is critical to your success in using this software.

Say you have just imported daily OHLCV bars for all the stocks in the S&P 500 index, as shown in **Tutorial 3**.

Imagine that data as sitting in RealTest's active memory, like a virtual Excel Workbook, with a separate Worksheet for each symbol, something like this:



	A	B	C	D	E	F	G	H	I
1	Date	Open	High	Low	Close	Volume			
2	6/19/2020	150.47	150.57	144.78	144.9	1,042,900			
3	6/18/2020	147.4	148.84	145.62	147.71	588,200			
4	6/17/2020	144.11	150.03	142.74	148.64	1,382,700			
5	6/16/2020	143.04	144.01	138.24	142.66	1,018,900			
6	6/15/2020	131.16	139.39	130.33	137.85	1,074,000			
7	6/12/2020	137.05	137.29	131.31	135.73	682,700			
8	6/11/2020	136.74	138.82	131.6	133.12	842,000			

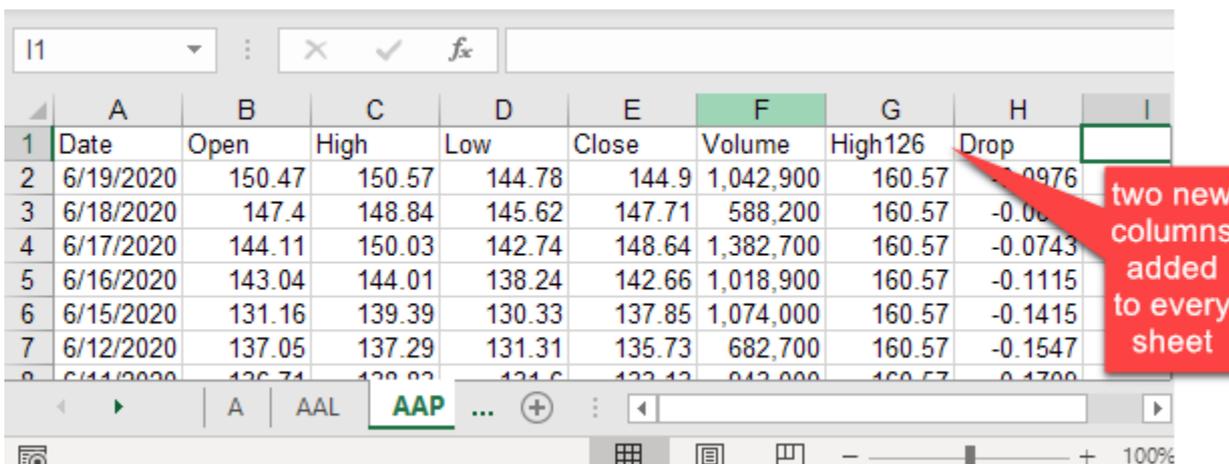
Now when you run a script that includes a Data section, here's what happens:

1. Every item defined under Data (in this case High126 and Drop) becomes a new column in EVERY worksheet in this imaginary workbook.
2. The formula provided for each data item is calculated for every bar of every stock and the resulting values are stored in the cells of that column.



**Data:**  
High126: Highest(C,126)  
Drop: C / High126 - 1

Now the imaginary workbook looks like this:



	A	B	C	D	E	F	G	H	I
1	Date	Open	High	Low	Close	Volume	High126	Drop	
2	6/19/2020	150.47	150.57	144.78	144.9	1,042,900	160.57	0.9976	
3	6/18/2020	147.4	148.84	145.62	147.71	588,200	160.57	-0.0000	
4	6/17/2020	144.11	150.03	142.74	148.64	1,382,700	160.57	-0.0743	
5	6/16/2020	143.04	144.01	138.24	142.66	1,018,900	160.57	-0.1115	
6	6/15/2020	131.16	139.39	130.33	137.85	1,074,000	160.57	-0.1415	
7	6/12/2020	137.05	137.29	131.31	135.73	682,700	160.57	-0.1547	
8	6/11/2020	136.74	138.82	131.6	133.12	842,000	160.57	-0.1700	

The two new columns, High126 and Drop, have been added to all 500 worksheets in this imaginary workbook, and for each of those worksheets, values have been calculated and stored for (in the tutorial example) all 1,628 rows of each of these two columns in each of these 500 worksheets. That's 1,628,000 calculations that were made and stored.

All these calculations happen nearly instantly when you run the example scan. In a more complicated multi-strategy trading system model, there will often be billions of calculations to make. Fortunately,

RealTest is very fast and takes advantage of all available CPU cores to do this job as quickly as possible.

As a further speed optimization, calculated Data items persist in memory as long as the same data file is in use and their formulas (or the formulas of other items that they refer to) have not been changed. So if, for example, you're running a number of backtests using the same underlying data calculations and only varying, for instance, the exit parameters, that set of tests can run extremely quickly.

To put it another way, calculated data columns are like global variable arrays in a standard programming language. As in an Excel worksheet, once a value has been calculated and stored, it can be referred to in any other formula.

Data column formulas can also refer to their own values that have already been calculated. See **Self-Referential Items** for details about how that works.

To look "under the hood" at the data columns currently residing in memory, use the **Debug Panel**.

Another special feature of the data section is the ability to calculate **cross-sectional** (breadth) data.

Because data columns are pre-calculated before a test is run, they are not allowed to access any test-specific context. If you have a need for intermediate formulas with test-specific context access, that is what the **Library Section** is for.

### 17.9.2.1. One-Pass Data Formulas

---

*The following material is kept here for reference and/or curiosity, but it is no longer necessary to understand and follow these guidelines.*

*Functions and indicators that can support one-pass calculation are now automatically calculated this way even when they're embedded within a larger formula expression.*

---

A regular Data Section item formula is calculated by looping through all the dates of the symbol and evaluating the formula for each date. If the formula includes an indicator or rolling bar function with a long lookback length, this can take some time to accomplish, since it has to "roll back" that many bars from each bar of each symbol.

For most types of formulas, this can't be avoided, and fortunately RealTest already performs these calculations as quickly as possible, using multiple threads.

For certain commonly-used functions and indicators, it is possible to dramatically speed up their calculation if they are structured and declared a certain way.

The difference in speed between the standard calculation method and "one-pass" is roughly a factor of one third of the lookback length. For example, a 100-day moving average can be calculated approximately 33 times faster when the one-pass optimization can be applied.

When you think about a backtest universe with 10,000 stocks going back 20 years (5000 daily bars per stock), you can start to imagine how much faster long-lookback indicators can be calculated if you structure your scripts to take advantage of one-pass mode.

All of the functions and indicators listed below will be automatically calculated in one pass if and only if the following conditions are all true:

- the indicator is the only element of the data item formula (not part of a larger expression)
- the length argument is a constant (literal number) or a parameter reference

For example, below are two ways to calculate simple uptrend / downtrend variables:

```

▼ Data:
// this will calculate the 50-day and 200-day averages the slow way
// it will redundantly calculate each one twice for every bar of every stock!
uptrend: ma(c,50) > ma(c,200)
downtrend: ma(c,50) < ma(c,200)
// calculation time for all Russell 3000 constituents since 2010: 50 seconds

▼ Data:
// this will calculate the two averages very quickly using one-pass
// and then look up their values for the uptrend/downtrend indicators
ma50: ma(C,50)
ma200: ma(c,200)
uptrend: ma50 > ma200
downtrend: ma50 < ma200
// calculation time for all Russell 3000 constituents since 2010: 1 second

```

In the first Data section above, RealTest cannot use one-pass calculation for either formula, because they each contain two functions and an operator.

In the second Data section, the moving averages are each given their own data items, which allows them to be calculated using one-pass.

Note the comments that compare the calculation speed (your results may vary depending on your hardware).

The following functions and indicators support one-pass calculation if they are specified as described above:

- ATR
- ADX
- MDI
- PDI
- RSI
- RSIF
- RRSI
- BBTOP
- BBBOT
- KBBOT
- KBTOP
- MACD
- MACDS
- MACDH
- SUM
- MA / AVG
- EMA / XAVG
- Highest / HHV
- Lowest / LLV
- SinceHigh
- SinceLow
- StdDev
- Skewness
- Kurtosis
- HVOL

Additionally, the following functions support one-pass calculation when their optional length or "nth" arguments are NOT present:

- CountTrue
- TrueInRow
- SinceTrue
- UntilTrue
- WhenTrue
- SumSince

It is worth your while to carefully structure your Data Section so that long-lookback indicators can take advantage of one-pass calculation!

To do this most effectively, look through all of your data and strategy formulas to find any of the above indicators, give each unique one its own data item, and change all the places where it is used to

references to that data item.

The **bensdorp\_book.rts example script** gives an excellent illustration of this technique. Imagine (or test, if you're curious) how much more slowly that set of seven strategies would run without the one-pass calculations!

## 17.9.2.2. Self-Referential Items

---

As well as being able to refer to prior **Data Section** items, the formula of an item can also *refer to itself*, as it is being calculated.

Use of self-referential data item formulas is an advanced programming technique that you will most likely never require. But if you do, this shows how they work.

Data item formula calculation occurs in a loop from earliest to latest bar of each stock. (This is analogous to how you would set up a data series calculation in an Excel column, by typing a formula in the first cell, then doing "fill down". Excel automatically adjusts each cell reference to produce a running calculation.)

As a first example, let's calculate and store an "all-time high" series for each stock.

A typical formula for all-time high would be:

```
▼ Data:
  ATH: Highest(H, BarNum)
```

This basically says "for each bar, go back through all the prior bars and find the highest high." Calculating this item this way would require 1 reference to the oldest bar, 2 references to the bar after it, 3 to the bar after that, and so on. For 10 years of data, which is about 2500 bars, calculating ATH this way would require  $2500 \cdot (2500 + 1) / 2 = 3,126,250$  bar references! (Thank you, Mr. Gauss.)

Instead, if you specify the formula like this

```
▼ Data:
  ATH: iif(BarNum=1, H, Max(H, ATH[1]))
```

then it only needs to compare two values for each bar: that bar's High, and the last calculated value of ATH. Doing it this way for 10 years of data would therefore require only 5,000 bar references, or about 1/625 as many as the first way.

---

Another example using this technique would be to calculate ATR using the original Welles Wilder formula as described in his book *New Concepts in Technical Trading Systems*. RealTest uses this same formula internally in the **ATR indicator**, so there's no need to calculate it yourself, but it's a fun example for this topic.

At the time that Wilder did most of his research, most traders did not have access even to a calculator, let alone a computer. He therefore favored exponential-style moving average calculation vs. using simple averages, mainly because they are so much faster to calculate by hand.

For example, ATR(14) can be calculated using this self-referential item formula:

```
▼ Data:
  atr14: iif(BarNum=1, TR, (atr14[1] * 13 + TR) / 14)
```

To generalize this to a parameterized ATR length, you could use:

```
▼ Parameters:
  len: 14

▼ Data:
  atrX: iif(BarNum=1, TR, (atrX[1] * (len-1) + TR) / len)
```

Similarly, this technique can be used to calculate your own EMA, though as with ATR, there is no reason to favor this approach over simply using the **EMA indicator**.

For example, EMA(20) can be calculated using this self-referential item formula:

```
▼ Data:
  ema20: iif(BarNum=1, C, ema20[1] * (1 - 2/21) + C * 2/21)
```

To generalize this to a parameterized EMA length, you could use:

```
▼ Parameters:
  len: 20

▼ Data:
  emaX: iif(BarNum=1, C, emaX[1] * (1 - 2/(len+1)) + C * 2/(len+1))
```

You may have noticed that the EMA formula is a bit more complex than the ATR one. This may be a clue as to why Wilder preferred his non-standard smoothing technique.

For more examples of self-referential items, see *flipper.rts* and *supertrend.rts* in the **Examples** folder.

### 17.9.2.3. Bar-Size-Specific Items

---

RealTest supports **multiple bar sizes** and makes it easy to mix them with correct date alignment.

Each script has a **BarSize** setting and can optionally specify a **Strategy** bar size that is different from the **Settings** bar size.

Each item defined in the **Data Section** is, by default, calculated and stored using the *Settings* bar size.

Data item arrays are *independent* of any strategy, therefore strategy bar sizes are not considered when calculating them.

If the settings bar size is *Daily* and the script includes a *Weekly* strategy, the data items used by that strategy must be explicitly written as weekly calculations.

There are two ways to make the bar size of a data item be different from the settings bar size:

1. Prefix the item name with a bar size name:

```
▼ Settings:
  BarSize: Daily

▼ Data:
  weeklyAvg20: Avg(C, 20)
```

2. Use Extern:

```
▼ Settings:
  BarSize: Daily

▼ Data:
  externAvg20: Extern(~Weekly, Avg(C, 20))
```

Both of the above will calculate the average of the past 20 weekly closing prices. The difference is in how these calculated values are stored.

In the first example, one value per week is stored in the data item's memory array.

In the second example, one value per day is stored (a new value each Friday which is repeated until the following Friday).

References to either of the above from other formulas will always return the same value.

This difference in how these items are stored can be seen by enabling this item in View / Program Options:

Program Options

Default File Paths

Scripts: C:\RealTest\Scripts

Data: C:\RealTest\Data

Output: C:\RealTest\Output

Data Formula Calculation

Maximum Memory (%): 95   31.8 GB installed

Maximum Threads: 6   6 cores available

Extract one-pass items  Auto #OnePerX  Log calculation details

Here is the output after calculating both items above for SPY from the start of 2015 through July 2023:

Name	BarSize	DataType	#OnePer	Values
weeklyAvg20	Weekly	price		904
externAvg20	Daily	price		4,354

The Extern method is preferable when calculating weekly or monthly data items for reference by a daily barsize strategy. Although more memory will be consumed for each such item, references to those items will be far faster. Each lookup of a non-daily Data item value corresponding to a daily bar takes time. Storing redundant daily copies of each non-daily bar value eliminates the need for those lookups.

The name prefix method is preferable when calculating weekly or monthly data items for reference in a strategy of that same barsize. In that case the opposite applies. It uses less memory and is more efficient to store those arrays in their own bar sizes when that's how they'll be referenced.

In summary, chose the other-barsize Data item style to match the context from which it will be referenced.

### 17.9.3. TestData Section

**TestData** is a specialized subset and variation of the **Data Section**.

To summarize their similarities and differences:

	Data Section	TestData Section
When is it calculated?	before any test starts, for all bars of all stocks	during each test, for one bar of all stocks at the start of each test date
How is it calculated?	optimized for speed using multiple threads, one-pass for many indicators, and only recalculating formulas that changed since last time	single-threaded, can't use one-pass, all items recalculated every test, but still quite fast if long-looback indicators are avoided
What can its formulas reference?	bar data, <i>Parameters</i> , previously calculated <i>Data</i> items (including itself), or <i>Library</i> items that only reference these things	everything that <i>Data</i> formulas can reference, anything that <i>Strategy</i> formulas can reference, and other <i>TestData</i> items (including itself)
Cross-sectional (breadth) support?	yes	yes
How is it stored?	in allocated memory arrays: 8 bytes per item * number of stocks * number of bars	ditto

Your *TestData* formulas are evaluated during each test at the start of processing for each test date (*BarSize* period). At each such interval, they are evaluated for only the current bar of every stock that has a bar for that date, with the value then stored in the *TestData* item's memory array.

As with *Data* items, *TestData* items can be referenced by name for the current bar, any past bar (using [n] offset notation) or as parameters to any multi-bar indicator.

*TestData* items can also **refer to themselves** in the same way *Data* items can.

As an example of how *TestData* is useful, consider a strategy like the **ndx\_rotate.rts** example where you want to incorporate a liquidity filter based on current **S.Equity** (account balance) as the test proceeds (assuming no withdrawals).

This example uses the **#rank** breadth operator to calculate its **posrank** variable:

```
▼ Data:
uptrend: c > Avg(C,200)
bullmkt: Extern($SPY, uptrend)
factor: 0.4 * PctChg(C,63) + 0.2 * PctChg(C,126) + 0.2 * PctChg(C,189) + 0.2 * PctChg(C,252)
canhold: InNDX and C > 10 and factor > 0 and uptrend and bullmkt
posrank: #rank canhold * factor
```

However, since *Data* is calculated before the test starts, you can't refer to *S.Equity* in that formula.

The solution is to move the last two *Data* items into a *TestData* section, and add a liquidity calculation:

```
▼ Data:
uptrend: c > Avg(C,200)
bullmkt: Extern($SPY, uptrend)
factor: 0.4 * PctChg(C,63) + 0.2 * PctChg(C,126) + 0.2 * PctChg(C,189) + 0.2 * PctChg(C,252)

▼ TestData:
liquid: S.Equity/positions < 0.001 * ma(c*v,20) // position size will be < 0.1% of average turnover
count: #sum InNDX and liquid
canhold: InNDX and C > 10 and factor > 0 and uptrend and bullmkt and liquid
posrank: #rank canhold * factor
```

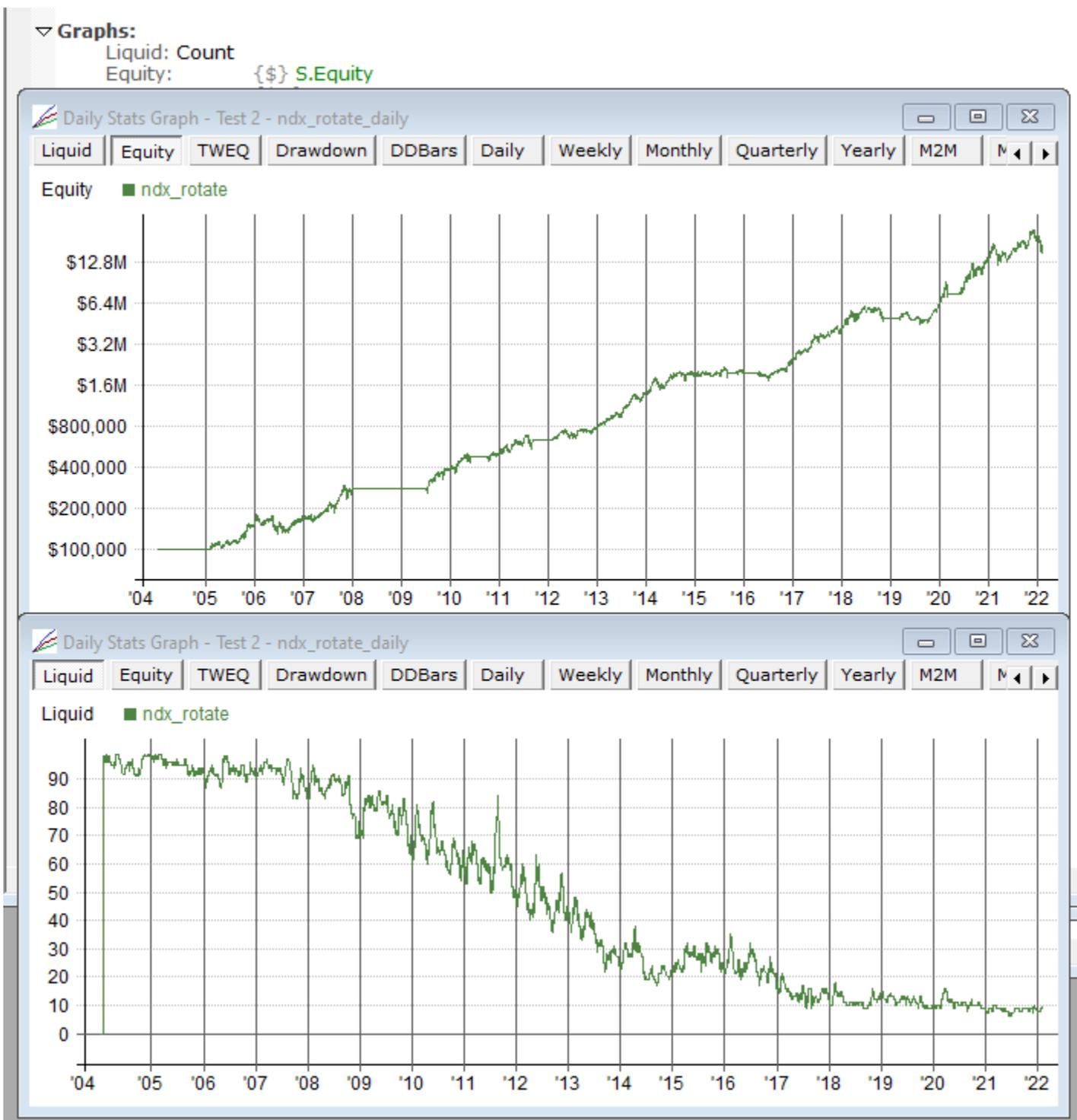
Note that in the above example, *S.Equity* is referenced directly. This works because there's only one strategy in the script.

A key point to remember, however, is that *TestData* is not strategy-specific. It is similar to **TestScan** in this regard. References to test stats, position information, etc. each have an implied **Combined** wrapper, i.e., the above *S.Equity* is interpreted as *Combined(S.Equity)*.

If this script contained other strategies, and you wanted to use only this strategy's equity value for your liquidity constraint, you would need to change the above to *Extern(@ndx\_rotate, S.Equity)*.

Notice that we've also added a *count* item, to make it easier to see the impact over time of this liquidity constraint.

By temporarily adding a **Graphs** item that displays *count*, we can see at a glance what the impact of this liquidity constraint would have been in this model:



In a \$3M+ account, if you applied this constraint, the "Nasdaq 100" would become the "Nasdaq 10" ...

## 17.9.4. Scan and TestScan Sections

The **Scan** section is, in a way, similar to the Data section. You might think of it as a data query and reporting tool.

The items provided under Scan are similar to Data items in that each provides a formula to be calculated. Each named formula becomes a column in the scan output.

The two exceptions are the **Filter** formula and the **Sort** specification.

The *filter* formula is evaluated for every bar of every stock to determine whether to include that bar in the scan output. A bar is included only if the Filter formula evaluates to "TRUE" or a non-zero value.

The *sort* specification lets you name the column(s) to use for initial sorting of the rows of the scan after it is generated.

Scans can be useful during research to find specific examples of a setup for further study, or in a production system to produce the candidate list for daily trading.

See **Tutorial 3** for a simple example of how to run a scan.

Here are the data and scan definitions from that example:

```
▽ Data:
  High126: Highest(C,126)
  Drop: C / High126 - 1

▽ Scan:
  Filter: C > 10 and Avg(V, 20) > 100000 and Drop < -0.25
  Price: {#2} C
  High126: High126
  Drop: {%2} Drop
```

Notice how the scan items are able to refer to the data items, and that it's fine to give a scan column the same name as a data item.

If a Scan item name begins with an underscore, that item will not be shown in the scan output. Rather, it will just serve as an intermediate variable that other items can refer to.

As with Results, Graphs, and Charts, Scan column formulas may include a **format specification comment** within their formulas.

Scans can be run for any date or a range of dates. The output is a new **scan window**.

The data file and date range to use when running a scan can be specified by adding a **ScanSettings section** to the script. If no settings are specified, the settings currently shown on the **Settings Panel** will be used.

See **Multi-Row Scan** for information on how to output more than one row for each symbol+date.

## TestScan

The TestScan section works the same as Scan except for these details:

- TestScan must be defined in a script that also includes at least one Strategy or Benchmark section
- Formulas within a TestScan definition may refer to any syntax element that a Strategy formula can refer to
- TestScan is run by checking Scan in the Test Output area of the Settings Panel or adding *TestOutput: Scan* to the **Settings** section of the script
- The date range of a TestScan is always confined to the one last bar of the test that was just run
- Other settings for a TestScan, such as ScanSaveAs, are defined in TestSettings rather than ScanSettings

The main purpose of a TestScan vs. a general-purpose Scan is to generate customized order lists. The feature was added to RealTest for this purpose, before the **CSV Order Baskets** capability was introduced.

See **Test Output Scan** for more details on how TestScan works.

## 17.9.5. Settings Sections

---

RealTest provides many settings which can be optionally specified in your scripts.

The **Settings** section is the place to specify settings that will apply in every run mode.

Only use the following run-mode-specific sections when you need different settings for different modes.

Use **ScanSettings** for settings to apply only apply when running a **Scan**.

Use **TestSettings** for settings to apply when running a **Test**, **TestScan**, or **Optimization**.

Use **OrderSettings** for settings to apply only when **generating orders**.

Run-mode-specific settings sections must appear **after** the general *Settings* section in a script if both are present.

Any setting in a run-mode-specific settings section will override the same setting in the general settings section when the script is being run in that particular mode.

Any setting in a run-mode-specific settings section that is not also in the general settings section will get its default value when the script is not run in that mode.

A few of these script-specifiable settings can also be specified interactively using the **Settings Panel**.

For those settings that appear on that panel, if the setting is not specified in the script then the value currently showing on the panel will be used.

When a script is running, the settings panel will temporarily display any settings that the script has specified that also appear on the panel.

When the run is finished, most settings panel settings are restored to their prior values. In other words, most script-specified settings do not persist in the user interface.

Exceptions to the above are: **DataFile** and **BarSize**. Those persist on the settings panel because **Data Section** arrays that persist in memory are dependent on those settings and would need to be recalculated if they were changed.

The following is a table of all available settings with their types and defaults:

<b>Setting</b>	<b>Type</b>	<b>Default</b>
<b>AccountSize</b>	dollars	settings panel
<b>BarSize</b>	Daily, Weekly, Monthly	settings panel
<b>Currency</b>	USD, EUR, GBP, CAD, AUD, etc.	none
<b>DataFile</b>	folder path and file name	settings panel
<b>EndDate</b>	date	settings panel
<b>ExchangeMap</b>	folder path and file name	none
<b>HolidayList</b>	folder path and file name	none
<b>KeepTrades</b>	see topic for choices	settings panel
<b>LegacyMode</b>	True or False	
<b>NumBars</b>	number	settings panel
<b>OrderClerkFolder</b>	folder path	none
<b>OrdersFile</b>	folder path and file name	none
<b>OrdersLiveData</b>	True or False	False
<b>OrdersMktAsLmtPct</b>	number	none
<b>OrdersMode</b>	see topic for choices	Text
<b>OrdersNetLiq</b>	folder path and file name	none
<b>OrdersTemplate</b>	folder path and file name	none
<b>RandomSeed</b>	number	none
<b>ResultsFile</b>	folder path and file name	none
<b>RiskFreeRateSym</b>	symbol (no leading \$)	none
<b>SaveChartsTo</b>	folder path	none
<b>SavePositionsAs</b>	folder path and file name	none
<b>SaveScanAs</b>	folder path and file name	none
<b>SaveStatsAs</b>	folder path and file name	none

<b>SaveTestListAs</b>	folder path and file name	none
<b>SaveTradesAs</b>	folder path and file name	none
<b>SaveTradesType</b>	Compact or Full	Full
<b>ScanNoDefCols</b>	True or False	False
<b>ScanNoHeader</b>	True or False	False
<b>ScanNoWindow</b>	True or False	False
<b>SkipTestIf</b>	formula (references parameters)	none
<b>StartDate</b>	date	settings panel
<b>StatsIncludeCash</b>	True or False	False
<b>SymChangeList</b>	folder path and file name	none
<b>TestName</b>	string	settings panel or script name
<b>TestOutput</b>	see topic for choices	settings panel
<b>TestScanAllDates</b>	True or False	False
<b>UseAvailableBars</b>	True or False	False

## 17.9.6. Library Section

---

The Library Section and the **Data Section** look very similar and can, to some degree, be used interchangeably. Their implementation and intended usage are, however, quite different.

The Data Section items are all pre-calculated before any test or scan is run. The calculated values are stored in memory arrays alongside the loaded bar data. Their calculation is extremely fast since it can take advantage of multi-threading, and they can be retrieved from memory very efficiently when accessed since they're stored as fixed-size arrays.

Library Section items, in contrast, are calculated "as needed". So if a formula refers to the same library item twice, it will be calculated twice. Another way to think of Library items is as "formula snippets". It is as if the library item's formula was copied and pasted into the formula that refers to it in place of its name.

While clearly less efficient than the Data Section approach, the advantage here is that a library formula has full access to the context in which it is used. For example, you cannot refer to something like **FillPrice** or **BarsHeld** or **S.Equity** in a Data item. No test has been run yet so there is no context to evaluate these.

In a library item, however, if it is referenced from a strategy formula, then every syntax element available to the strategy is available to the library formula. In other words, library formulas always inherit the context of the formula that refers to them.

See the **mr\_sample\_debug.rts example script** for an example of using the Library section.

### Library Functions

Library formulas can optionally serve as general-purpose one-line functions.

A Library item can be referenced with up to 9 arguments, as if it were a function, e.g. *my\_library\_item(high, low)*.

Library items that serve as functions can access the values passed to them by referencing the special build-in variables **Arg1 - Arg9**.

Here is a contrived example to show how this works:

Library:  
AvgOfThree: (Arg1 + Arg2 + Arg3) / 3

Data:  
TypicalPrice: AvgOfThree(H, L, C)

As with any built-in function, the arguments passed to a library function can be any expression (including other library functions).

## 17.9.7. Strategy Section

A script can include any number of Strategy sections. All strategies found in a script will be included when a test is run.

To exclude a strategy from a test, temporarily comment it out. This can quickly be done by putting the cursor in the strategy header line, then pressing Ctrl+/ or selecting "Comment Selection" from the **Script Menu**.

... or click here to toggle ...

... resulting in ...

```
//Strategy: mr_long // mean-reversion long strategy
// Using: base
// Side: Long
// EntrySetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMA5) // oversold
// EntryLimit: LongLimit
// ExitLimit: FillPrice * (1 + Target/100) // intraday profit target
// ExitRule: select(C > C[1], "up day", BarsHeld == 5, "time stop") // sell on first up day or after 5 days

Strategy: mr_short // mean-reversion short strategy (gets DESTROYED in January 2021 by the meme stocks)
Using: base
Side: Short
EntrySetup: Universe and C > (1 + PctExt / 100) * Max(O, C[1], EMA5) // overbought
EntryLimit: ShortLimit
ExitLimit: FillPrice * (1 - Target/100) // intraday profit target
ExitRule: select(C < C[1], "down day", BarsHeld == 5, "time stop") // cover on first down day or after 5 days
```

Do the same in reverse to remove the comments and include the strategy again.

Another way to toggle comment in/out is to click in the left margin adjacent to the line or section to be toggled, as shown above.

### 17.9.7.1. Strategy Names

---

Each strategy can optionally be named, by typing a name (up to 40 characters) after the "Strategy:" marker.



```

▽ Strategy: mr_long ← mean-reversion long strategy
  Using: base
  Side: Long
  EntrySetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMaX) // oversold
  EntryLimit: C - ATRx // drops another ATR
  ExitRule: C > C[1] or BarsHeld > 5 // sell on first up day or after 5 days

▽ Strategy: mr_short ← mean-reversion short strategy
  Using: base
  Side: Short
  EntrySetup: Universe and C > (1 + PctExt / 100) * Max(O, C[1], EMaX) // overbought
  EntryLimit: C + ATRx // rises another ATR
  ExitRule: C < C[1] or BarsHeld > 5 // cover on first down day or after 5 days

```

The name you provide will be used in all references to the strategy in graphs, trade lists, etc.

If you omit the names, strategies are named "Strategy 1", "Strategy 2", etc.

### 17.9.7.2. Special Strategy Types

---

There are two alternative ways to define a **Strategy** -- as a **Benchmark** or as a **Template**. All three of these strategy types are defined the same way and support all the same elements. The only difference is in which keyword is used to declare them. For an example of a script that uses all three types, see `mr_sample_benchmark.rts`.

A **Benchmark** strategy is run and its stats are calculated exactly as if it was a regular strategy. It has its own set of stats and will appear among the strategies shown in stats graphs. The only difference is that the stats of a benchmark are not included in the combined results. One use of a benchmark strategy is to plot a simple buy-and-hold equity line for visual comparison with your own strategy. A more advanced usage would be to define a basic version of a strategy which the actual strategy can refer to.

**Template** strategies serve to prevent the need to copy and paste elements that are common to several strategies in a script. Template strategies are not "run" in a backtest. In older versions of RealTest, there was a single template strategy called "Defaults". This has now been replaced with support for multiple named templates.

To define a template strategy, simply declare it as, for example, "Template: base". To include the elements from that template in another strategy (or template or benchmark), add the statement "Using: base" to the strategy. To include more than one template, list them separated by commas, as in "Using: base1, base2, base3".

When a strategy uses a template, it inherits all the elements defined in that template (and any others that the template uses). It can then optionally replace some of those elements by declaring them within the strategy.

In addition to the above, **StatsGroup** is also a strategy-like script element. Like these others, *StatsGroup* defines an outer-level section. However, the only *Strategy* elements that *StatsGroup* is allowed to contain are: **Using**, **Allocation**, **MaxEntries**, **MaxExposure**, **MaxInvestment**, **MaxPositions**, **MaxSameCat**, and **MaxSetups**.

The original purpose of *StatsGroup* was to define a combined statistics series, similar to *Combined*

(which is automatically added to any test with more than one Strategy) but with a smaller subset of strategies involved in the combination.

For example, say you had two long and two short strategies in a system, called long1, long2, short1 and short2. By default your stats would include 5 series (5 lines on the Equity graph etc.): long1, long2, short1, short2, and Combined.

If you also wanted to see combined stats for both long strategies and both short strategies, you could simply add these statements to the script:

```
▽ StatsGroup: Longs  
  Using: long1, long2  
▽ StatsGroup: Shorts  
  Using: short1, short2
```

This defines two new stats series, Longs and Shorts, calculated by combining trades from both strategies of the group into its stats.

In addition to using *StatsGroup* to add more stats series to your test output, you can also reference them from your strategy logic. Just as the ongoing stats of any *Strategy* or *Benchmark* can be accessed using *Extern(@strategyName, S.xxx)*, all of the *StatsGroup* series that you define can similarly be accessed using **Extern**.

The default *Allocation* for a *StatsGroup* is *Combined(S.Equity)*. Allocation is only relevant in this context because of its use in the calculation of percent-based results statistics such as **S.TWEQ**, **S.MaxDDPct** and **S.NetPct**. Only override the default Allocation if you need these percentages to be calculated differently.

With the addition of top-down mode in RealTest 2.0.26, the *StatsGroup* and *Combined* functionality was expanded. In this new mode a *StatsGroup* can include any of the **Max...** items listed above.

An additional section called **Combined** can optionally be added explicitly. If present, it also supports all of the *Max...* items that *StatsGroup* does. Note that *Combined* as a strategy-like section is the only such section that cannot be named, and it also may not contain *Using* or *Allocation* definitions.

When **Max...** constraints are specified for a *StatsGroup* or *Combined*, they are applied to all strategies in that group together.

For example, if *MaxExposure: 100* is defined for *Combined*: then a setup in any strategy will be skipped if adding that position would push the combined exposure above 100%.

Similarly, these constraints in a *StatsGroup* ensure that setups will be skipped if entering those positions would exceed any them.

See Also: **Backtest Engine Details** and **Capacity Constraints**

### 17.9.7.3. Strategy Elements

---

The elements within each strategy definition represent all the available inputs to the general-purpose backtesting engine of RealTest. A few of them are required, the rest are optional (with obvious defaults).

To see a quick list of supported elements, press F2 with the cursor in a blank space within a strategy definition (but not at the start of a line):

▼ Strategy: **temp**

- **Allocation:** -- denominator for position size and stats calculations (dollars)
- **Ambiguity:** -- assumption when order of entry/target/stop is ambiguous (Neither by default)
- **BarSize:** -- size of bars to use in this strategy
- **CalendarSym:** -- symbol with strategy's market date list (\$ not required)
- **CashInOut:** -- cash deposited or withdrawn (dollars)
- **CashList:** -- path of cash in-out list CSV file
- **Category:** -- position category value (number)
- **CloseSlip:** -- slippage for market-at-close fills (dollars per share)
- **Commission:** -- commission (dollars per share)
- **Compounded:** -- overrides automatic determination of whether a strategy compounds its equity
- **DebugEntry:** -- debug log output at Entry Setup/Limit/Stop evaluation time
- **DebugExit:** -- debug log output at Exit Rule evaluation time
- **DebugTargetStop:** -- debug log output at Exit Limit/Stop evaluation time
- **EntryLimit:** -- entry limit price
- **EntryScore:** -- entry fill score value for EntryRank (highest assumed to fill first)
- **EntrySetup:** -- entry setup condition
- **EntrySkip:** -- entry skip condition
- **EntryStop:** -- entry stop price
- **EntryTime:** -- entry time constant
- **EntryTradeValue:** -- value to store in T.ValueIn at entry time
- **ExitLimit:** -- exit target price
- **ExitLimitQty:** -- number of shares to exit (for partial ExitLimit exits)
- **ExitLimitTime:** -- exit limit time constant
- **ExitQty:** -- number of shares to exit (for partial ExitRule exits)
- **ExitRule:** -- exit condition
- **ExitScore:** -- position score value for ExitRank (for use in ExitRule)
- **ExitStop:** -- exit stop price
- **ExitStopQty:** -- number of shares to exit (for partial ExitStop exits)
- **ExitStopTime:** -- exit stop time constant
- **ExitTime:** -- exit rule time constant
- **ExitTradeValue:** -- value to store in T.ValueOut at exit time
- **HolidayList:** -- strategy-specific holiday list file (only needed for orders)
- **LimitExtra:** -- extra excursion needed for limit to fill (dollars per share)
- **LimitSlip:** -- slippage for fills at a limit price (dollars per share)
- **MarkToMarket:** -- whether to add mark-to-market value of open positions to strategy equity
- **MaxEntries:** -- maximum entry fills per day
- **MaxExposure:** -- maximum open investment exposure (percent)
- **MaxInvested:** -- maximum open investment amount (dollars)
- **MaxNewExp:** -- maximum new investment exposure (percent)
- **MaxNewInv:** -- maximum new investment amount (dollars)

Another way to think of these strategy elements is to imagine that rather than using scripts, RealTest had a strategy definition dialog box.

Some of the elements below require a constant, such as "Long" vs. "Short", or "ThisClose" vs. "NextOpen".

Other elements require a formula that evaluates to true (1) or false (0), and others take a formula that evaluates to a number.

See **Strategy Element Value Types and Defaults** for a table of element types.

The following are more detailed descriptions of each strategy element.

### 17.9.7.4. Strategy Element Value Types and Defaults

The following table lists each of the above strategy elements along with its value type and default (what happens if this element is not included in a strategy):

Element Name	Element Type	Value Type	Default Value
<b>Allocation</b>	formula	dollars	full account size
<b>Ambiguity</b>	constant	<i>Default, Stop, Target, Neither</i>	<i>Neither</i>
<b>BarSize</b>	constant	<i>Daily, Weekly, Monthly</i>	<i>Daily</i>
<b>CalendarSym</b>	constant	symbol (no leading \$)	none
<b>CashInOut</b>	formula	dollars	0

<b>CashList</b>	file	file path	none
<b>Category</b>	formula	number	0 (none)
<b>CloseSlip</b>	formula	dollars per share	0 (use slippage)
<b>Commission</b>	formula	dollars	0 (none)
<b>Compounded</b>	constant	True or False	inferred from Quantity
<b>DebugEntry</b>	formula	string or number	none
<b>DebugExit</b>	formula	string or number	none
<b>DebugTargetStop</b>	formula	string or number	none
<b>EntryLimit</b>	formula	price	0 (none)
<b>EntryScore</b>	formula	number (highest first)	alphabetical by symbol
<b>EntrySetup</b>	formula	condition	0 (false)
<b>EntrySkip</b>	formula	condition	0 (false)
<b>EntryStop</b>	formula	price	0 (none)
<b>EntryTime</b>	constant	logical time of day*	<i>NextOpen</i>
<b>EntryTradeValue</b>	formula	number	0
<b>ExitLimit</b>	formula	price	0 (none)
<b>ExitQty</b>	formula	shares	entire position
<b>ExitLimitTime</b>	constant	logical time of day*	<i>Intraday</i>
<b>ExitLimitQty</b>	formula	shares	entire position
<b>ExitRule</b>	formula	condition	0 (false)
<b>ExitStop</b>	formula	price	0 (none)
<b>ExitStopQty</b>	formula	shares	entire position
<b>ExitStopTime</b>	constant	logical time of day*	<i>Intraday</i>
<b>ExitTime</b>	constant	logical time of day*	<i>NextOpen</i>
<b>ExitTradeValue</b>	formula	number	0
<b>HolidayList</b>	file	file path	none
<b>LimitExtra</b>	formula	dollars per share	0 (none)
<b>LimitSlip</b>	formula	dollars per share	0 (use Slippage)
<b>MarkToMarket</b>	constant	True or False	True
<b>MaxEntries</b>	formula	count	unlimited
<b>MaxExposure</b>	formula	percentage	unlimited
<b>MaxInvested</b>	formula	dollars	unlimited
<b>MaxNewExp</b>	formula	percentage	unlimited
<b>MaxNewInv</b>	formula	dollars	unlimited
<b>MaxNewPos</b>	formula	count	unlimited
<b>MaxPerTurn</b>	formula	count	1
<b>MaxPositions</b>	formula	count	unlimited
<b>MaxSameCat</b>	formula	count	unlimited
<b>MaxSameSym</b>	formula	count	1 per strategy
<b>MaxSetups</b>	formula	count	unlimited
<b>OpenSlip</b>	formula	dollars per share	0 (use slippage)
<b>OrderMktAsLmtPct</b>	formula	number	0
<b>OrderNote</b>	formula	number or string	none
<b>OrdersFile</b>	constant	file path	none
<b>PriceRound</b>	formula	number	tick size
<b>QtyFinal</b>	formula	shares or contracts	Quantity formula value
<b>QtyPrice</b>	constant	OrderPrice or FillPrice	OrderPrice in default mode
<b>QtyRound</b>	formula	number	1
<b>QtyType</b>	constant	<i>Shares, Value, Percent</i>	<i>Shares</i>

<b>Quantity</b>	formula	shares or contracts	account size / entry price
<b>Reduce</b>	constant	True or False	False
<b>SetupScore</b>	formula	number (highest first)	alphabetical by symbol
<b>SetupSkip</b>	formula	condition	0 (false)
<b>Side</b>	constant	<i>Long,Short,Both</i>	<i>Both</i>
<b>Slippage</b>	formula	dollars per share	0 (none)
<b>StopSlip</b>	formula	dollars per share	0 (use Slippage)
<b>StrategyScore</b>	formula	number (highest first)	script sequence
<b>TLAdjusted</b>	constant	True or False	False
<b>TLDateFmt</b>	constant	MDY or DMY	program options setting
<b>TLFields</b>	constant	ordered list of field names	none
<b>TLIgnoreRules</b>	constant	True or False	False
<b>TLStratName</b>	constant	string	current strategy name
<b>TLTimeShift</b>	constant	number of hours	0
<b>TradeList</b>	file	file path	no trade list
<b>Using</b>	special	strategy name(s)	not using anything

\* For the above elements that specify "logical time of day" as their *Value Type*, available constant values are *ThisClose*, *Intraday*, *NextOpen*, or *NextClose*. Use the links to the corresponding *Element Name* topics for further information.

## 17.9.8. Parameters Section

---

Parameters or optimization variables are defined in their own section of the script. They can be given any name and used by any strategy.

Parameters can also be referred to in the formulas for items in the Data Section.

The syntax for defining parameter variables is different from other script sections where the standard formula syntax is used.

The end product of a parameter variable declaration is a list of values to use.

The syntax supports all of the following methods of defining a value list:

- A single number
- Traditional loop statement, e.g. "from 5 to 50 step 5"
- Loop with multiplication, e.g. "from 5 to 200 mult 1.1"
- Optionally, either of the above can include a "def" and/or "round" clause, e.g. "from 5 to 50 step 5 def 20" or "from 5 to 200 mult 1.1 round 1"
- Or you can simply list the values you want, separated by commas

Note that all values used in this section must be simple numeric constants. No expression syntax (e.g. division) is allowed.

The following are examples of supported syntax:

```

▼ Parameters:
foo: 12
bar: 24
opt1: 1,2,3,5,8,13,21,34,55,89,144,233
opt2: from 1 to 10
opt3: from 0.5 to 1.5 step 0.1
opt4: from 1 to 10 def 5
opt5: from 5 to 250 mult 1.2 round 1

```

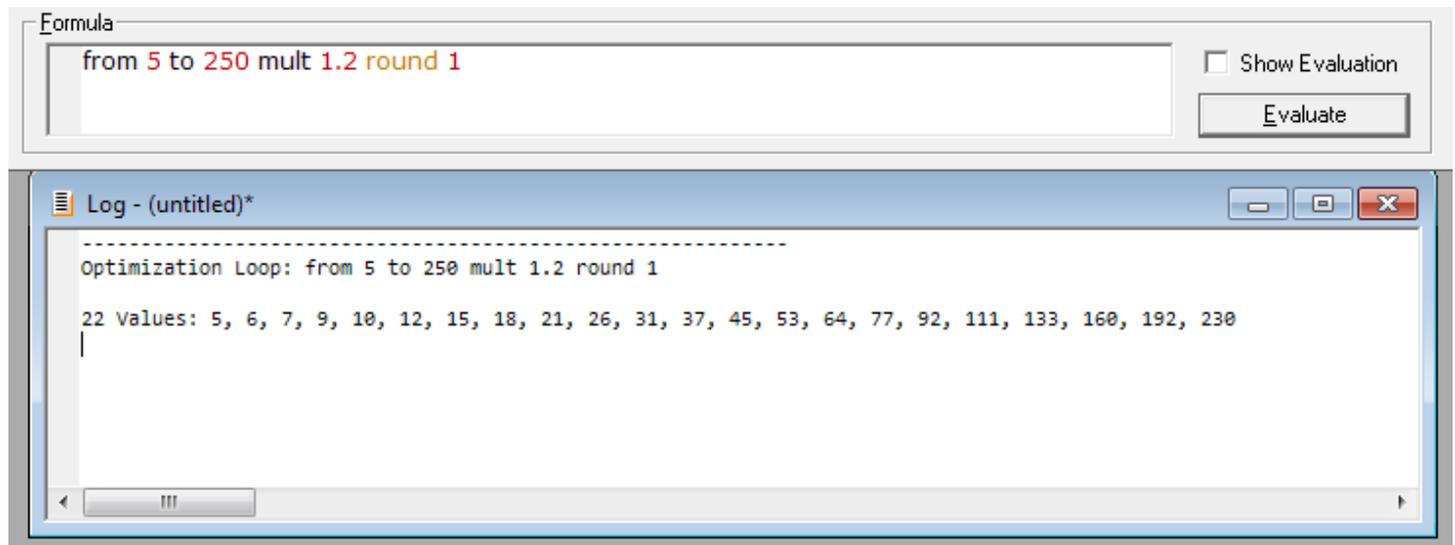
If no default ("def") is specified, then the default value will be the first number in a list or loop.

Default values are used in the following circumstances:

- a single test is run, rather than an optimization
- an optimization is run but that parameter is not selected in the optimization parameter list
- a formula from a non-strategy section, e.g. Scan, refers to a Data item that refers to a parameter in its formula

The **Debug Panel** can be used to experiment with the above syntax.

Evaluating any "from" statement in the debug formula causes the resulting value list to be written to the log window.



## 17.9.9. Results Section

---

The Results Section is where the columns to display in any **Results Window** are defined.

Here is the default set of Results window columns and their definitions:

```

Active Script - C:\REALTEST\Results.rts
Results:
  Periods:      {#} S.Number
  NetProfit:    {$0} S.Equity - S.StartEquity
  _CAR:         {%2} (S.Equity / S.StartEquity)^(1/(S.Number / S.BPY)) - 1
  _AAR:         {%2} ((S.Equity / S.StartEquity) - 1) / (S.Number / S.BPY)
  ROR:          {%2|} iif(S.Compounding, _CAR, _AAR)
  MaxDD:        {%2|} -S.MaxDDPct
  // Ratio:     {#2} ROR / -MAXDD
  Exits:         {"Trades"} Sum(S.Exits,S.Number)
  _wins:        {#} Sum(S.Wins,S.Number)
  _losses:       {#} Sum(S.Losses,S.Number)
  PctWins:      {%2} _wins/Exits
  AvgWin:       {%2} Sum(S.WinPct,S.Number) / _wins
  AvgLoss:      {%2} Sum(S.LossPct,S.Number) / _losses
  WinLen:       {#2} Sum(S.WinBars,S.Number) / _wins
  LossLen:      {#2} Sum(S.LossBars,S.Number) / _losses
  Expectancy:   {%2} Sum(S.NetPct,S.Number) / Exits
  // AvgWinDlr: {$2} Sum(S.WinDlr,S.Number) / _wins
  // AvgLossDlr: {$2} Sum(S.LossDlr,S.Number) / _losses
  // ExpectDlr:  {$2} Sum(S.NetDlr,S.Number) / Exits
  ProfitFactor: {#2} Sum(S.WinDlr,S.Number) / Sum(S.LossDlr,S.Number)
  Sharpe:       {#2} SQR(252)*Avg(S.NetPct,S.Number)/StdDev(S.NetPct,S.Number)
  // Sortino:   {#2} SQR(252)*Avg(S.NetPct, S.Number) / StdDev(Min(0,S.NetPct), S.Number)
  // Skipped:   {%2} (1-Exits/Sum(S.Setups,S.Number))
  // Exposure:  {%2} Avg(S.Exposure / S.Alloc,S.Number)
  Usage:        {%2} Avg(S.Usage / S.Alloc, S.Number)

```

You are welcome to edit this file (press F9 when viewing a results window) and/or include a different set of column definitions in other test scripts.

Examples of some columns that you might want to add are provided as **comments** in this default script (rows beginning with //).

Every test result record contains the same set of underlying **Test Statistics Arrays** (the green items in the above example). Your column formulas select which stats you want to display, allow you to specify the format of the numbers, and make it possible to calculate higher-level stats however you prefer.

Results column formulas are calculated only once, at the end of a test run. When they are calculated, the context is the last date of the test.

This is why, for example, **S.Number** is used to display the number of periods (dates) over which a test was run. The daily stats record for the first date is 1, the second date is 2, and so on. At the end of the above test, there were 6,894 stats records, so the number of the last record was 6,894.

Similarly, for stats where we want the total count of something for the entire test, we use the **Sum** function.

Results formulas can optionally reference **Trade Statistics Functions**, though calculating these can be slow for tests with very high trade counts.

A **format specification comment** can be included in any item. If no format is specified, the item will use default number formatting.

The vertical bar `{|}` in a Results item format specification indicates that this item should be calculated after each day of the test and displayed in the status bar as the test is running.

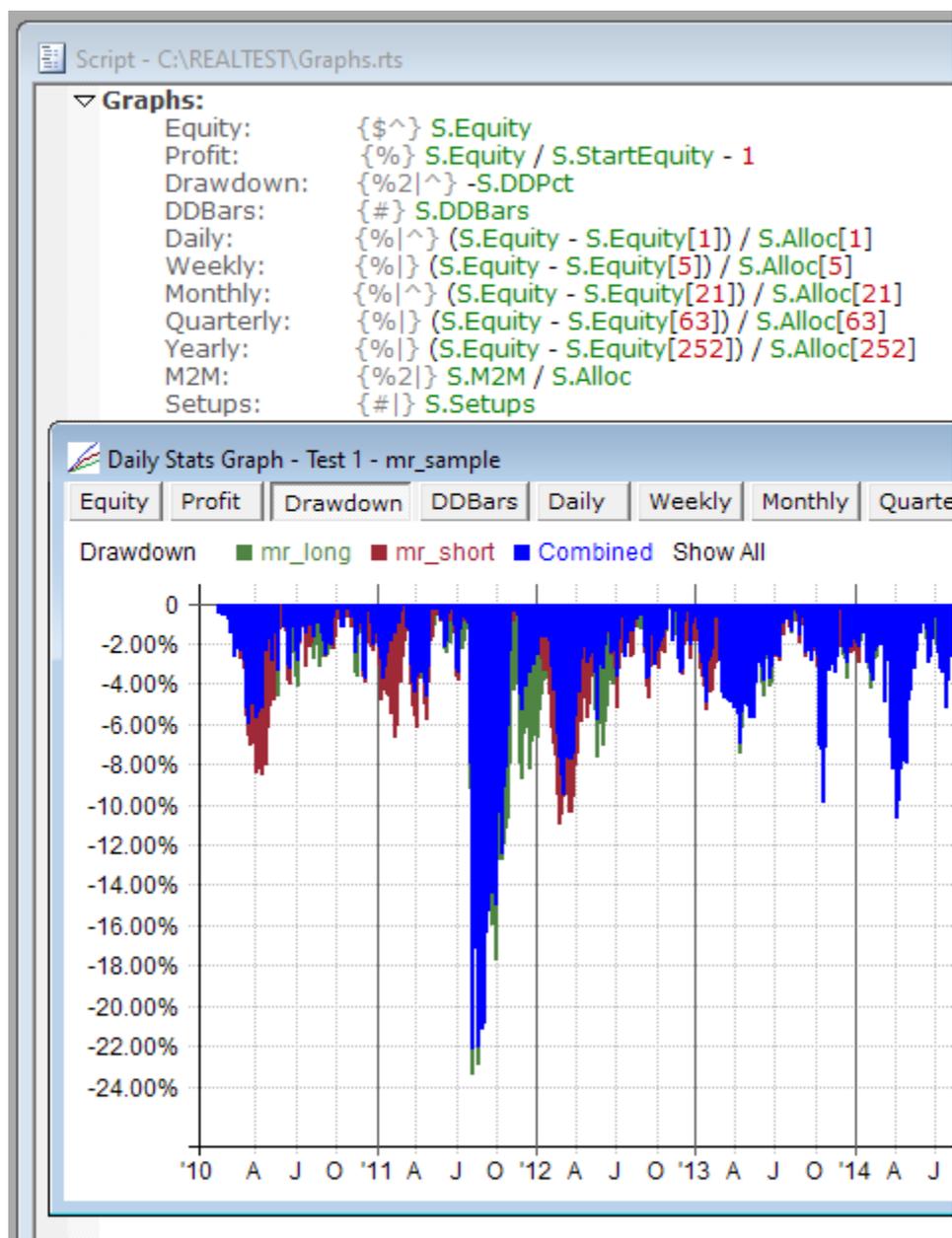
To apply Results section formulas to all currently open Results windows, press F4 or click Apply.

## 17.9.10. Graphs Section

This script section defines each graph to include in the list of available daily stats graphs for any test that has been run.

If the active script does not include a *Graphs* section then the default graphs script *graphs.rts* is used.

Each graph type defined in this section becomes a button on the tab bar at the top of every **Graph Window**.



To access the underlying formulas for any Results, Graphs, Trades or Chart window, press the F9 key or use the context menu.

Each item defined in the *Graphs* section defines a graph to include. The item name becomes the name of the graph, and the item formula is evaluated for every date in the test results record and then plotted as either a continuous line or a histogram.

If a graph item name begins with an underscore, that item will not be plotted. Rather, it will just serve as an intermediate variable that other items can refer to. This can be useful to avoid repeating the same expression several times or to calculate an indicator that requires several steps.

A **format specification comment** can be included in any item and has two purposes. If no format is specified, the item will be graphed as a line using default number formatting.

The vertical bar `{|}` is used to specify that a histogram is to be drawn, otherwise a line graph will be drawn.

If a number format is specified, it will be used when drawing the Y axis and whenever a value is displayed for a specific date.

The special format code `{^}` is used to specify which graphs are included in **Test Summary Reports**.

To apply Graphs section contents to all currently open Graph windows, press F4 or click Apply.

## 17.9.11. Trades Section

The *Trades Section* is where the columns to display in any **Trade List Window** are defined.

Here is an example of a script (the default trades.RTS) which adds some custom columns:

```

Active Script - C:\REALTEST\Trades.rts
Trades:
  Bars: T.Bars
  PctGain: {%-2} T.Points / T.PriceIn
  Profit: {$-2} T.Profit
  _UP: T.Highest - T.PriceIn
  _DN: T.PriceIn - T.Lowest
  Fraction: {%-2} T.Fraction
  _MFE: iif(T.Side = 1, _UP, _DN)
  PctMFE: {%-2} _MFE / T.PriceIn
  _MAE: -iif(T.Side = 1, _DN, _UP)
  PctMAE: {%-2} _MAE / T.PriceIn
  Comm: {$2} T.CommIn + T.CommOut
  Slip: {$2} T.SlipIn + T.SlipOut
  Div: {$-2} T.Div

// Dollar-based MFE/MAE (for non-compounded models)
// DlrMFE: {$2} _MFE * T.QtyIn * T.PtVal
// DlrMAE: {$2} _MAE * T.QtyIn * T.PtVal

// Other useful columns
// Size: T.PriceIn * T.QtyIn {$0}
// Points: {#2} T.Points
  
```

Applying this script causes the new columns to be shown in all open trade windows.

Here is how the trade list looks when scrolled horizontally to show the custom columns:

DateOut	TimeOut	QtyOut	PriceOut	Reason	PctGain	Profit	UP	DN	MFE	PctMFE	MAE	PctMAE	Comm	Slip	Div
4/19/94	close	2,156	44.36	exit rule	-4.35%	(\$3,078.12)	1.94	3.03	1.94	4.18%	-3.03	-6.54%	0	0	\$1,267.73
9/29/98	close	2,060	104.94	exit rule	123.05%	\$130,965.94	72.19	2.44	72.19	153.44%	-2.44	-5.18%	0	0	\$11,711.10
11/4/99	close	1,924	136.53	exit rule	15.31%	\$37,737.34	23.84	4.66	23.84	20.14%	-4.66	-3.93%	0	0	\$2,864.84

The formulas used in the Trades section will most often refer to syntax elements that begin with T.

When a Trade List Window is opened, RealTest loops through all the trade records that are embedded in the results record for that test and calculates each of the formulas in the Trades section for that trade. The context of any T. variable is therefore the specific trade being evaluated in this loop.

Trade List formulas can also refer to any stock bar elements such as Open, High, Low, Close and also any Data section elements. The context when bar/data elements are referenced will always be the bar on which the trade was **exited**. To refer to the **entry** bar of a trade, use **T.Bars** as an offset. For example, the close of the entry bar will be **C[T.Bars]**. To refer to the **EntrySetup** bar (the last completed bar at entry time), use **C[T.Bars+1]** unless **EntryTime** was *ThisClose* (market-on-close).

A **format specification comment** can be included in any item. If no format is specified, the item will use default number formatting.

To apply Trades section formulas to all currently open Trade List windows, press F4 or click Apply.

**Literal strings** or string functions such as Format can also be used in custom trade list columns.

In addition to the above, there are two special Trades items that can optionally be added: **Filter** and **Sort**.

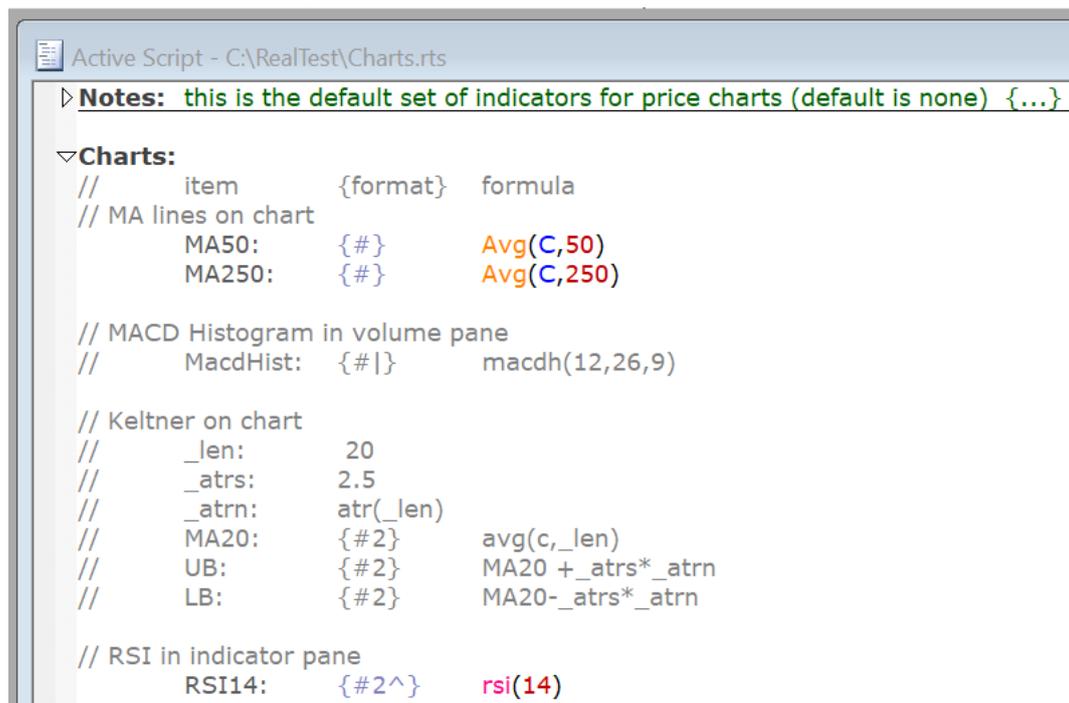
The *filter* formula is evaluated for each trade to determine whether to include it in the list. A trade is included only if the Filter formula evaluates to "TRUE" or a non-zero value.

The *sort* specification lets you name the column(s) to use for initial sorting of the rows of the trade list

after it is generated.

## 17.9.12. Charts Section

This script section defines any lines to be plotted on bar or candlestick charts whenever they are viewed. If the active script does not include a **Charts** section then the default charts script *charts.rts* is applied.



```
Active Script - C:\RealTest\Charts.rts
> Notes: this is the default set of indicators for price charts (default is none) {...}
▼ Charts:
//      item      {format}  formula
// MA lines on chart
//      MA50:      {#}      Avg(C,50)
//      MA250:     {#}      Avg(C,250)

// MACD Histogram in volume pane
//      MacdHist:  {#|}     macdh(12,26,9)

// Keltner on chart
//      _len:      20
//      _atrs:     2.5
//      _atr:      atr(_len)
//      MA20:      {#2}     avg(c,_len)
//      UB:        {#2}     MA20 +_atrs*_atr
//      LB:        {#2}     MA20-_atrs*_atr

// RSI in indicator pane
//      RSI14:     {#2^}    rsi(14)
```

Each item defined in the *Charts* section defines a data series line to display. The item name becomes the name of the line on the chart, and the item formula is evaluated for every bar and then plotted as a continuous line.

If a Charts item name begins with an underscore, that item will not be plotted. Rather, it will just serve as an intermediate variable that other items can refer to. This can be useful to avoid repeating the same expression several times or to calculate an indicator that requires several steps.

A **format specification comment** can be included in any item and has multiple purposes. If no format is specified, the item will be plotted along with the price bars and use their scale.

The accent {^} is used to plot this line in the top (indicator) pane rather than the price pane. (You may have to press 'i' or open the chart options dialog to show the indicator pane.)

The vertical bar {|} is used to plot this line in the bottom (volume) pane rather than the price pane. (You may have to press 'v' or open the chart options dialog to show the volume pane.) If no lower pane indicators are specified, the pane will show volume bars. (It is not currently possible to plot both indicators and volume bars in the lower pane.)

If a number format is specified, it will be used whenever the value of the item is displayed for a specific bar.

To apply Charts section contents to all currently open Chart windows, press F4 or click Apply in the Tool Bar.

Items defined above are plotted below.



If a special Charts item called "Text" is defined, it is interpreted as a string formula and its output is appended to the chart's legend row.

The example belows shows how to add the current bar's "volume surge" to the chart legend using this technique:

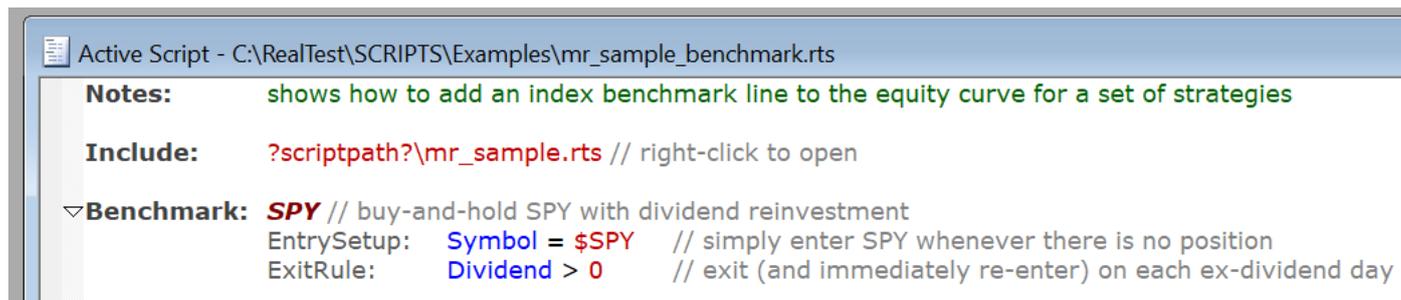


## 17.9.13. Include Section

Though this appears as an outer-level "script section", **Include** is just a simple statement to allow a script to include other scripts. This capability is provided to make it easier to maintain common

elements that several scripts share. Examples would be common **Data Section** items, a frequently used **Import** definition, or even a **Strategy** (e.g. an index benchmark) that you often add to other strategies in a script.

Several of the **Example Scripts** that are based on **mr\_sample.rts** illustrate the use of Include:



```
Active Script - C:\RealTest\SCRIPTS\Examples\mr_sample_benchmark.rts
Notes:      shows how to add an index benchmark line to the equity curve for a set of strategies
Include:    ?scriptpath?\mr_sample.rts // right-click to open
▼Benchmark: SPY // buy-and-hold SPY with dividend reinvestment
EntrySetup: Symbol = $SPY // simply enter SPY whenever there is no position
ExitRule:   Dividend > 0 // exit (and immediately re-enter) on each ex-dividend day
```

This example includes the entire original script and then adds a SPY benchmark to it.

Note that the script being included must consist of one or more entire script sections.

For example, it would not work to combine a set of strategy entry formulas from one script with exit formulas for a different script by including both scripts. It would, however, work to include a Template section that various strategies share.

You can combine items of the same section if that section supports multiple instances. This is the case for Data, Library and Parameters, so Include can be used to share common subsets of those sections among various scripts if desired.

Finally, if you have certain elements that you want to include in every script in a folder, RealTest supports automatic inclusion of any script named **autoinclude.rts** in the same folder as the script being run. If this special script is found, it is included at the top of any other script from the same folder.

## 17.10. Formula Syntax

---

Many elements of a RealTest script allow a formula expression to be provided to calculate the value to be used.

Examples of places where formulas are allowed include:

- all the items in the **Data**, **Scan**, **Results**, **Graphs**, **Trades** and **Charts** sections
- most of the elements of the **Strategy** section (see the **Strategy Element Table** for specifics)
- the **ExcludeIf** element of the **Import** section
- the **SkipTestIf** and **TestName** elements of the **Settings** section
- the **Debug Panel**
- the **Plot Options** dialog

RealTest formula syntax is straightforward and will seem familiar if you've used any other testing or scanning software, or ever written an Excel cell formula.

All expressions (including rolling time-series functions) are fully recursive (able to be nested), placing no limits on the type of any argument.

Every term in every function (even **bar offset** specifications) can be its own formula.

For example, RealTest includes both weighted moving average (WAvG) and Hull moving average (HAvG) built-in functions in addition to the more common simple (MA) and exponential (EMA) flavors.

With the WAvG function and recursive syntax, it was not really necessary to add the built-in HAvG

(nevertheless it is there).

The two formulas below will produce the same result:

HAvg\_A: HAvg(C, 20)

HAvg\_B: WAvg((2 \* WAvg(C, 20 / 2) - WAvg(C, 20)), SQR(20))

As an alternative to writing deeply nested expressions, you can use the Data section to calculate various parts of a complex rule, and then refer to them by name in other formulas, as shown in many of the **Example Scripts**.

If a formula cannot be evaluated, the return value will be "nan" (not a number).

The only reasons that a formula cannot be evaluated are:

- not enough bars were available to fulfill its lookback plus offset length (unless **UseAvailableBars** was specified)
- it refers to a specific symbol or strategy that is not available in the current data file

Once a formula has returned nan, all other formulas that contain or reference it will also be nan.

If desired, you can use IsNan function to see if a formula would return nan, and/or the NoNan function to force a formula to return 0 instead of nan.

**Data Section** formulas that return nan will store nan as that data item, so all references to it will be nan.

**Scan** or **Trades** Section formulas that return nan will display that item as "nan".

**Strategy Element** formulas that return nan are treated the same as formulas that return 0, so in general you don't need to worry about nan in your strategy formulas.

## 17.10.1. Operators

---

The following table lists all of the operators can be used in any RealTest script formula.

The general structure of a formula is *term operator term*.

The definition of *term* in this structure is ... any formula.

The examples in the table below show the simplest possible structure, with just a number on either side of each operator.

Operator	Alternative	Precedence Rank	Description	Example
()		1	parentheses	(1+1)*2 is 4
+ -		2	unary plus or minus	+2 + -2 is 0
NOT	!	2	logical not	NOT false is true
^		3	power, as in x^y (x to the power y)	9 ^ 2 is 81
*		4	multiplication	9 * 2 is 18
/		4	division	6 / 2 is 3
MOD	%	4	modulo (remainder after division)	10 MOD 3 is 1
+		5	addition	2 + 2 is 4
-		5	subtraction	9 - 2 is 7
>		6	greater than	7 > 5 is true
>=		6	greater than or equal to	7 >= 7 is true
<		6	less than	7 < 5 is false

<=		6	less than or equal to	7 <= 7 is true
=	==	7	is equal to	7 = 7 is true
<>	!=	7	not equal to	7 <> 7 is false
AND	&&	8	logical and	true AND false is false
OR		9	logical or	true OR false is true

The *Precedence Rank* column comes into play when formulas include more than one operator. Lower numbers mean higher precedence.

Operators with higher precedence rank are evaluated first, no matter where they are in the formula.

Operators with the same precedence rank are evaluated from left to right.

The best practice, to avoid confusion, is to use parentheses to make precedence explicit in your formulas.

A couple of examples to clarify this:

- "value1 + value2 \* value3" would be the same as "value1 + (value2 \* value3)", because \* has higher rank than +.
- "condition1 OR condition2 AND condition3" would be the same as "condition1 OR (condition2 AND condition3)" because AND has higher rank than OR.

Special note about division:

RealTest formulas allow division by zero. Anything divided by zero is zero. Though this is mathematically incorrect, it removes the unnecessary complexity of having to check that every divisor is non-zero in your own formulas.

## 17.10.2. Formula Evaluation

---

The RealTest formula evaluator works like a "virtual machine". Each formula in the script is compiled to a binary format and preprocessed. The repeated evaluation that occurs during a test is therefore as efficient as possible.

One way in which evaluation is optimized is known as "short circuit". If it becomes known before evaluation is finished that there is only one possible result, the remainder of the evaluation process is skipped.

For example, in the pseudo-expression "0 and this and that and the other thing", nothing after the 0 would be evaluated, because it is logically impossible for an expression (or sub-expression) with "0 and ..." to be anything other than 0. Ditto with "1 or ...", "0 \* ...", "0 / ...", etc.

You can see this short-circuit in action if you experiment with some formulas in **Debug Panel** with "show evaluation" checked.

You can then use this knowledge of how formulas are evaluated to make your tests run even faster, by putting the most-likely-to-be-false terms at the beginning of your conditional formulas.

The best place to take advantage of short-circuit efficiency is in the **Data** section and in your **EntrySetup** formula.

For example, say you have a multi-part EntrySetup concept, such as:

- price between 10 and 80
- 20-day average volume at least 100K
- price above its 200-day moving average
- price down 3 days in a row

You could write this as:

```
C > MA(C,200) and MA(V,20) >= 100000 and CountTrue(C < C[1], 3) == 3 and C > 10 and C < 80
```

or you could write the same logic as:

```
C > 10 and C < 80 and CountTrue(C < C[1], 3) == 3 and MA(V,20) >= 100000 and C > MA(C,200)
```

In the first example, the 200-day moving average will have to be calculated for every stock in your database for every date in your backtest. Each of these calculations will require going back 200 bars from the current bar, adding all the closes, and then dividing the sum by 200. RealTest can actually perform these millions of lookups and calculations remarkably quickly, but your tests will run a lot faster if you write your formulas like the second example.

In the second example, formula elements are written in order of calculation speed. Because of short-circuit optimization, all of the preceding comparisons will have to be true in order for it to remain necessary to calculate the 200-day moving average. You can make an educated guess about which formula elements take the longest to calculate from the lookback length - how many bars back it must go to calculate its value for each bar.

### 17.10.3. Breadth Tags / Cross-Sectional Functions

---

RealTest makes it easy to specify and calculate cross-sectional or "breadth" functions in the **Data Section of a script**.

Whereas the usual rolling bar functions such as Avg, Sum, Highest, Lowest, etc. operate "vertically" (down a column of the same symbol for different dates), the breadth functions operate "horizontally" (across a row of the same date for different symbols).

The format of a data item that calculates a breadth function is:

*name: #function formula*

*Name* is the name of the data item, as usual.

*Function* is the specific breadth function to use.

*Formula* is the formula to calculate for each cell (a specific symbol on a specific date).

Only one breadth function can be used in a data item, and it must appear at the start of the formula for that item.

Any number of data items can have breadth tags, though, so if you need to perform a multi-level breadth calculation, just use multiple data items.

When the item is calculated for each date, the following things occur:

1. *formula* is calculated for every symbol with date for that date
2. *function* is calculated for the set of values produced by step 1
3. the result is stored as the value of the data item for every symbol for that date

Depending on which *function* is used, the end result may be the same for every symbol on a given date.

**#Avg**, **#Count**, **#Highest**, **#Lowest**, **#Median**, **#StdDev** and **#Sum** will, by definition, all produce the same answer for every cell in a row of data.

Don't worry, *formula* will still only be evaluated once per symbol per date, and *function* is only calculated once per date. The point is that this result is then separately stored for every symbol for that date. While this might sound silly, it actually simplifies other formulas that want to access this result, because they can simply refer to it in the context of their own current symbol.

**#PercentRank** and **#Rank**, on the other hand, will (by definition) produce a different answer for every cell in a row of data. In this case, *formula* is still only evaluated once per symbol per date, after which *function* is calculated by sorting the resulting array of values and then storing each symbol's ordinal (expressed as rank percent or rank number) as its data value for that date.

The **example script** `ibd_rs.rts` shows how to use this feature to calculate IBD-style "relative strength" ranking.

Here is how the S&P 500 components were ranked for IBD RS on 6/26/20:

The screenshot shows an Active Script window titled "Active Script - C:\REALTEST\Examples\ibd\_rs.rts". The code is as follows:

```
// compute IBD RS rankings for a universe of stocks

▽ ScanSettings:
  DataFile: SPX.rtd // see import_spx.rts for how to create this file
  EndDate: Latest
  NumDays: 1

▽ Data:
  strength: 0.4 * PctChg(C,63) + 0.2 * PctChg(C,126)
           + 0.2 * PctChg(C,189) + 0.2 * PctChg(C,252)
  srank: #Rank strength
  scount: #Count strength
  rs: Round(100 * (scount - srank) / scount, 0.01, 0)

▽ Scan:
  Strength: strength {#}
  RS_Rank: srank {#}
  RS: rs {#}
```

Below the script is a window titled "Scan - 505 Items" showing a table of results:

Date	Symbol	Strength	RS_Rank	RS
6/26/20	DXCM	97.50	1	99.80
6/26/20	NVDA	74.75	2	99.60
6/26/20	REGN	69.84	3	99.40
6/26/20	PYPL	67.00	4	99.20
6/26/20	WST	59.67	5	99.01
6/26/20	ADSK	50.42	6	98.81
6/26/20	FTNT	49.37	7	98.61
6/26/20	AAPL	49.07	8	98.41
6/26/20	EBAY	47.58	9	98.21
6/26/20	NOW	46.07	10	98.01

Note that if a breadth function formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the ranking list (reducing the total count) and its rank value would be **nan** as well. Similarly, **nan** values would not be included in the value count for any breadth functions that use it (e.g. #Avg, #Median, #StdDev).

## 17.10.4. External Symbol or Strategy Reference

The **Extern** function makes it easy to refer to data for a symbol other than the current one, or to access the stats of a different strategy (or the combined stats) within a test.

The syntax is: `Extern(item, expression)`

By default, the context of any expression is the current stock and the current strategy. This function simply creates a temporary context with a different symbol or strategy and evaluates your expression using that temporary context.

The syntax for an external symbol reference is: `$symbol`, e.g. `$MSFT`.

If a data file is currently loaded in memory while the script is being edited, then the editor's auto-completion mechanism will present a list of possible symbols as soon as you type the \$.

The syntax for an external strategy reference is: `@strategy_name`, e.g. `@mr_Long`

The special name `@combined` can be used to refer to the combined stats of all strategies.

Since this is a common requirement, an alternative and slightly shorter way to do this is to use the

*Combined(expression)* function, which is equivalent to *Extern(@combined, expression)*.

Once a script has been parsed at least once, the list of strategies that it contains are available for auto-completion after typing the @ sign.

Besides accessing the data of an external symbol or the stats of an external strategy, the *Extern* function is also useful within a multi-strategy system to access overall current position information.

The simplest example of when you'd use *Extern* is the specific symbol reference. For instance, to use an index ETF relative to its moving average as part of your **EntrySetup** rule, you'd say something like this:

```
EntryRule: Extern($SPY, C > MA(C,200)) and {the rest of your entry logic}
```

Actually, to avoid needlessly re-calculating the SPY 200-day average for every symbol every day, I'd implement this as follows:

```
▼ Data:
  Above200: Symbol==$SPY and C > MA(C,200)

▼ Strategy: your_strategy
  EntryRule: Extern($SPY, Above200) and {the rest of your entry logic}
```

The script **mr\_sample\_hedged.rts** in the **example scripts** directory shows a sophisticated example of how to use external strategy references to build a dynamic SPY hedge for a long/short system:

```
▼ Strategy: hedge // use SPY to hedge any overnight long/short imbalance (data file must include SPY)
  EntrySetup: Symbol = $SPY
  ExitRule: (Extern(@mr_short,S.Positions) - Extern(@mr_long, S.Positions)) <>
            (Extern(@mr_short,S.Positions[1]) - Extern(@mr_long, S.Positions[1]))
  Quantity: Hedge * (S.Alloc / NumPos) * (Extern(@mr_short,S.Positions) - Extern(@mr_long, S.Positions)) / C
```

While this looks complex, all it's doing is:

- calculate the difference between the number of positions in the other two strategies for today vs. yesterday
- if that difference has changed, then exit the former hedge and enter a new hedge based on the new difference

This implementation takes advantage of the following features of RealTest strategies:

- By default a position is only entered when there is currently no position in that symbol for that strategy, which is why **EntrySetup** needs no logic other than that the current symbol is SPY
- If the **Side** of a strategy is not specified, it is inferred from the sign of the **Quantity** calculation result.

Admittedly, in real trading you would not exit the entire hedge and re-enter at the new size, you'd just buy or sell the net change in shares required. But for backtesting purposes this works fine.

## 17.10.5. Special Syntax for Individual Futures Contract Testing

---

This category of external item reference is somewhat more specialized.

If you import all of the individual historical contracts for a futures market, you can, of course, refer to any specific contract by symbol, but how can you model something like a realistic roll-over strategy?

To solve this problem, RealTest does some extra processing on futures symbols during import.

To take advantage of this feature your futures symbols, if your data is not from **Norgate**, must have the same format used by Norgate.

Specifically, the format must be: XX-YYYYM, e.g. ES-2016Z, GC-2021K, etc.

By using the logic inherent in this naming convention, RealTest is able to construct a "symbol chain" during import, which it can later use to permit REALTIVE external symbol references.

The syntax for a relative reference is `Extern(&n, expression)`, where "n" is a numeric offset in either direction. Positive directions refer to newer contracts (ones that expire farther in the future) and negative directions to older ones (ones that expire sooner).

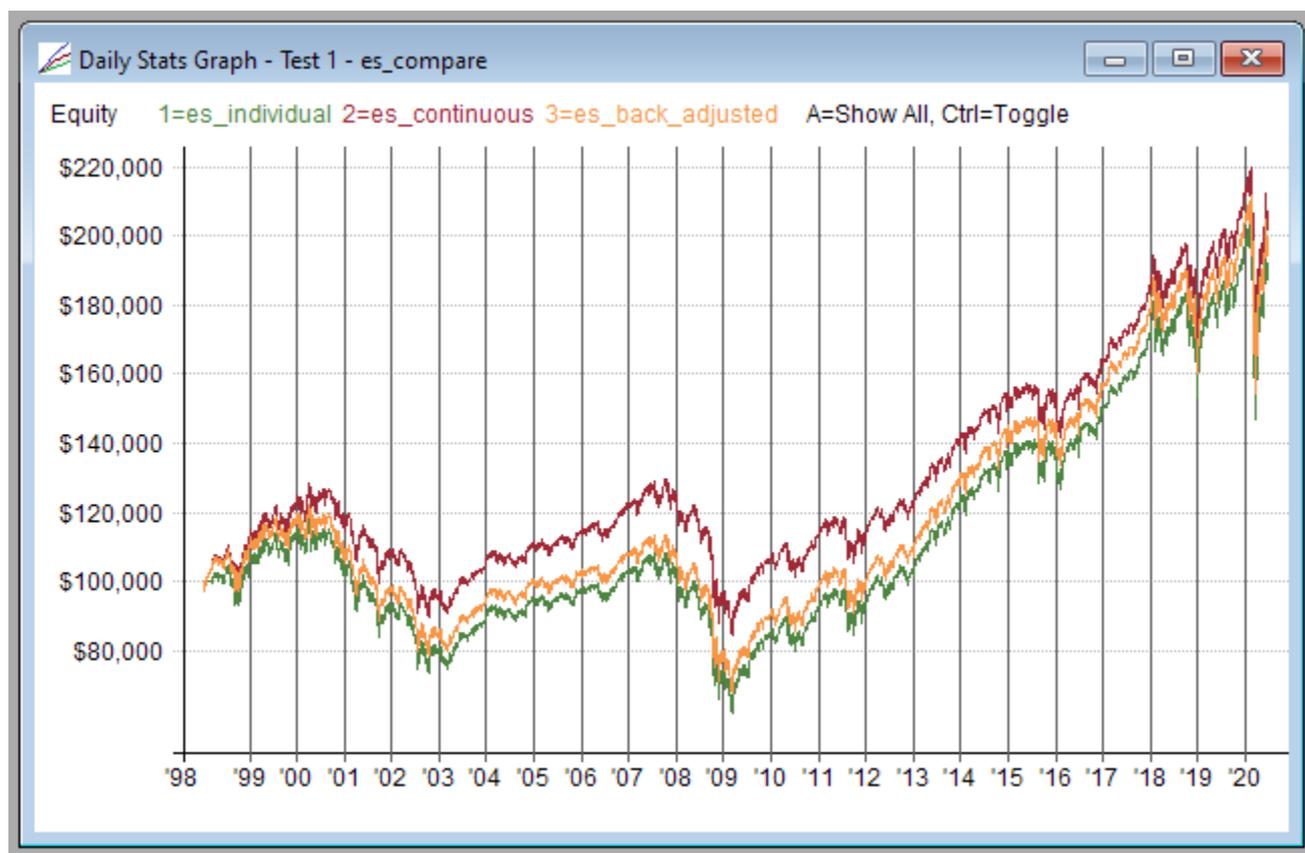
For example, if your current symbol is ES-2016Z, then `Extern(&1, C)` would give you today's close for ES-2017H, and `Extern(&-1,C)` would give today's close for ES-2016U.

The simplest application of this syntax is to accurately model "buy and hold" of a futures market using any desired rollover rule. For example, this strategy will hold 1 ES and roll at the open after the first day where the new contract has more volume than the old one:

```
▼ Strategy: es_individual  
EntrySetup: v > extern(&-1,V)  
ExitRule: v < extern(&1, V)
```

You can also use this technique to model spread trading, calculate contango/backwardation indicators, or who knows what else.

The script `es_compare.rts` in the **example scripts** directory includes the above strategy and runs a comparison study of modeling "buy and hold" ES since its inception using (1) individual contracts with accurate rollover transaction modeling, (2) a single continuous price series without back-adjustment, and (3) a single continuous back-adjusted price series.



When this kind of futures symbol chain exists, RealTest can also calculate market-specific breadth values such as ranking by volume the active contracts within each market on a given date. See **#ByMkt** and the `futures_calendar_spread.rts` and `futures_volume_rank.rts` example scripts for details.

Besides relative contract lookup, there is one additional variation of this syntax.

If your data file includes the **Norgate** continuous contract series as well as individual contract symbols, you can reference the corresponding continuous series for the current individual symbol by using either **Extern(&cc, formula)** if you want the non-back-adjusted one, or **Extern(&ccb, formula)** if you want the back-adjusted one.

The `futures_volume_rank.rts` example script shows how this technique can be used to implement a trend-following strategy that trades individual contracts while using a continuous series to generate its signals.

Going the other way, if the current symbol is a Norgate continuous contract symbol, the corresponding

individual contract with the nearest upcoming expiration date can be referenced using **Extern(&0, formula)**. The second nearest is **Extern(&1, formula)**, and so on.

Knowing this, it becomes simple to calculate spreads:

```
Library:
Spread: Extern(&cc, Extern(&0, C) - Extern(&1, C))
```

The **futures\_calendar\_spread.rts** example implements this in a slightly more efficient way, but the result is the same.

## 17.10.6. Statistics Values in Formulas

---

RealTest provides a large set of daily test stat variables which can either be used directly or combined in expressions to calculate any conceivable system metric.

In order to give each of these stat items the most logical name without preventing that same name from being used as a column variable in Results or Graphs definitions, the built-in items all have names beginning with "S."

Stats variables are used in all of the formulas in RESULTS.RTS and GRAPHS.RTS - the default definitions for all **Results** and **Graph** windows.

Stats variables can also be used in any formula-based **Strategy Element** definition. When a test is being run, each day's daily stats variables are calculated and stored at the end of that day during the test.

This makes it possible, for example, for **EntrySetup** or **ExitRule** or **Quantity** formulas to include references to the strategy's current results so far.

A simple reference to a stat variable returns the value for the current day only. To obtain summary stats, use multi-bar functions with the stat variables.

For example, say a test is on day (bar) 100 and you want to know the overall profit factor so far. Profit factor is defined as dollars won / dollars lost. The RealTest syntax for dollars won is **S.WinDlr** and for dollars lost is **S.LossDlr** (both return positive values). The number of stat days so far in a test is available as **S.Number**. The profit-factor-so-far formula would therefore be  $Sum(S.WinDlr, S.Number) / Sum(S.LossDlr, S.Number)$ . Or you might want to use a rolling 3-month profit factor, which would be  $Sum(S.WinDlr, 63) / Sum(S.LossDlr, 63)$ .

The following stat properties are exceptions to the above rule and are accumulated internally, such that each day's value incorporates the entire test so far:

**S.MinEquity, S.MaxEquity, S.MaxDDdlr, S.MaxDDPct, S.MaxDDBars**

(It should be clear from the Min/Max prefixes that this is how these work.)

To make all of the above more concrete, do the following:

1. Run a test or open a results file
2. Open the debug panel
3. Select a test number and strategy name under "Stats Context"
4. Select a date that is within the range of dates for the test (and not a weekend or holiday)
5. Click "Stats" under "Examine"

The output will look something like the following image. There are more columns in the stats window than are visible here. Scroll right to see the rest of them.

Data Context: Date: 7/28/17 Symbol: Chart

Stats Context: Test: 0001:mr\_sample Strategy: Combined Graph

Examine: Days Back: 50 Data Stats

mr\_sample.rts (untitled) @Combined Stats X

Active Script - C:\REALTEST\RELEASE\Examples\mr\_sample.rts

Results - (untitled)

Test	Notes	Dates	NumPos	PctExt	Periods	NetProfit	CAR	MaxDD	Sharpe	Sortino
0001	mr_sample	1/5/10 - 7/28/17	10	2.5	1,905	\$861,171	34.90%	-14.41%	2.29	4.04

Stats - Test 1 (mr\_sample) - Strategy @Combined

S.Positions	S.Exposure	S.M2M	S.MAE	S.MFE	S.StartEquity	S.MaxEquity	S.MinEquity	S.Equity
1	(\$98,105)	\$650	(\$1,503)	\$23,954	\$100,000	\$998,794	\$92,232	\$998,144
4	\$394,040	\$5,262	(\$6,606)	\$18,917	\$100,000	\$987,901	\$92,232	\$982,639
2	\$197,112	\$3,798	(\$4,335)	\$11,959	\$100,000	\$986,437	\$92,232	\$982,639
1	(\$95,659)	(\$1,295)	(\$2,618)	\$3,659	\$100,000	\$982,463	\$92,232	\$983,758
2	\$316	\$1,580	(\$3,413)	\$46,230	\$100,000	\$982,071	\$92,232	\$980,491
8	\$381,993	\$16,005	(\$36,804)	\$48,941	\$100,000	\$974,896	\$92,232	\$958,891
6	(\$365,812)	(\$14,333)	(\$30,029)	\$53,629	\$100,000	\$931,023	\$92,232	\$945,356
6	(\$365,621)	\$1,689	(\$9,445)	\$27,325	\$100,000	\$922,552	\$92,232	\$916,665
0	0	0	(\$9,027)	\$8,609	\$100,000	\$922,552	\$92,232	\$916,665
3	(\$273,910)	(\$2,604)	(\$10,417)	\$11,750	\$100,000	\$922,552	\$92,232	\$915,726
2	(\$181,478)	(\$5,194)	(\$12,406)	\$10,726	\$100,000	\$922,552	\$92,232	\$917,615
6	(\$544,550)	(\$938)	(\$10,052)	\$22,306	\$100,000	\$922,552	\$92,232	\$914,720
1	\$91,266	\$4,365	(\$7,418)	\$12,550	\$100,000	\$922,552	\$92,232	\$909,269
2	\$1,809	(\$2,314)	(\$8,867)	\$5,569	\$100,000	\$922,552	\$92,232	\$912,087
1	(\$89,616)	(\$4,983)	(\$20,324)	\$1,132	\$100,000	\$922,552	\$92,232	\$912,087
3	(\$269,999)	(\$14,387)	(\$24,726)	\$9,463	\$100,000	\$922,552	\$92,232	\$913,568
5	(\$450,921)	(\$13,684)	(\$25,351)	\$19,367	\$100,000	\$922,552	\$92,232	\$907,273
3	(\$269,156)	(\$977)	(\$4,526)	\$13,859	\$100,000	\$922,552	\$92,232	\$906,367
3	(\$89,287)	\$5,712	(\$24,769)	\$33,125	\$100,000	\$922,552	\$92,232	\$901,175
4	(\$354,576)	(\$10,670)	(\$22,747)	\$15,636	\$100,000	\$922,552	\$92,232	\$898,243
3	(\$264,942)	(\$6,227)	(\$12,452)	\$16,008	\$100,000	\$922,552	\$92,232	\$892,396
3	\$88,392	\$2,726	(\$6,810)	\$19,321	\$100,000	\$922,552	\$92,232	\$886,095

## 17.11. Syntax Element Categories

All of the elements of the RealTest Script Language syntax are listed here according to usage categories, alphabetically within each category.

### 17.11.1. Script Sections

A script can contain a variety of top-level sections. Specific section types are listed below.

- **Benchmark** - begin a new benchmark strategy definition
- **Charts** - chart indicator definitions
- **Data** - named formulas calculated once and stored in memory arrays before tests are run
- **Graphs** - graph type definitions
- **Import** - data import definitions
- **Include** - allows a script to always include another script
- **Library** - named formulas calculated when they are referenced, using the current context

- **Notes** - a free-form section in which to organize any notes about the script
- **OrderSettings** - override general settings when script is run as *Orders*
- **OrdersInclude** - allows a script to include another script only when being run as *Orders*
- **Parameters** - system parameter definitions
- **Results** - results column definitions
- **Scan** - filter and column definitions for general-purpose scans
- **ScanInclude** - allows a script to include another script only when being run as *Scan*
- **ScanSettings** - override general settings when script is run as *Scan*
- **Settings** - runtime settings for all script run modes
- **Strategy** - begin a new trading strategy definition
- **Template** - begin a new template strategy definition
- **TestData** - named formulas calculated for each bar as tests are run and stored in memory arrays
- **TestInclude** - allows a script to include another script only when being run as *Test*
- **TestScan** - filter and column definitions for scans that include trade and position information
- **TestSettings** - override general settings when script is run as *Test*
- **Trades** - trade list column definitions
- **WalkForward** - system parameter values by date

## 17.11.2. Settings

---

These define the settings to apply when running a script.

- **AccountSize** - starting capital amount
- **BarSize** - default test data timeframe
- **CashIntPct** - interest rate received for positive daily excess cash
- **Currency** - account base currency for multi-currency system models
- **DataFile** - test data file path
- **EndDate** - last date of test
- **ExchangeMap** - details about specific exchanges for which you may want to generate orders
- **HolidayList** - list of holidays (allows EndOfWeek or EndOfMonth to work on the last bar of data)
- **KeepTrades** - types of trades to store in each results record
- **LegacyMode** - activates the older (pre-2.0.26) way of processing setups and applying constraints
- **MarginIntPct** - interest rate charged for negative daily excess cash (margin loan)
- **NumBars** - number of market dates to test
- **OrderClerkFolder** - path of folder to use with OrderClerk
- **OrdersFile** - path\name of order list file to generate
- **OrdersLiveData** - allows order generation for "ThisClose" entry and exit times

- **OrdersMode** - specifies the format of generated orders
- **OrdersNetLiq** - path\name of a text file containing the current live Net Liquidation Value of a brokerage account
- **OrdersTemplate** - path\name of CSV order list template file
- **RandomSeed** - provides a way to use the same sequence of random numbers every time a script is run
- **ResultsFile** - path\name of RTR file to open or create before running a test
- **RiskFreeRateSym** - symbol of data series to store in the test statistics for later use when calculating Sharpe
- **SaveChartsTo** - path of folder in which to automatically save a chart for every row of the scan
- **SavePositionsAs** - path\name of CSV file to create at end of a test to list open positions
- **SaveScanAs** - path\name of CSV file to create and write scan output to
- **SaveStatsAs** - path\name of CSV file to create and write stats details to
- **SaveTestListAs** - path\name of CSV file to create with the list of test results as they appear in the results window
- **SaveTradesAs** - path\name of CSV file to create and write the trade list to
- **SaveTradesType** - format to use for *SaveTradesAs* output file
- **ScanNoDefCols** - allows the default Date and Symbol columns to be optionally omitted
- **ScanNoHeader** - allows creation of a CSV file with no header row
- **ScanNoWindow** - allows scanning directly to CSV without displaying the output in a window
- **SkipTestIf** - allows tests to be skipped in multi-parameter optimizations (e.g. useless parameter combinations)
- **StartDate** - first date of test
- **StatsIncludeCash** - whether to include cash deposits and withdrawals in percent-based test stats
- **SymChangeList** - path\name of CSV file containing list of symbol changes to use when processing imported trades
- **TestName** - give the test a name
- **TestOutput** - additional output and actions during and after a test
- **TestScanAllDates** - allows TestScan to output rows for every date of a test, not just the last date
- **UseAvailableBars** - allows simple averages and indicators to optionally be calculated with fewer bars than specified

### 17.11.3. Import Specification

---

All the information needed to import data for use in scans and tests.

- **Adjustment** - Norgate data adjustment type
- **CIIFamily** - Norgate corresponding industry index family
- **CIILevel** - Norgate corresponding industry index level
- **Classification** - Scheme to use when querying Norgate for sector and industry

- **Constituency** - Norgate index constituency symbol list
- **CSVDateFmt** - CSV date format (M/D/Y vs. D/M/Y if ambiguous)
- **CSVDelim** - CSV column delimiter
- **CSVFields** - CSV field order (comma-separated list)
- **CSVFile** - CSV data file path for for single-file import
- **CSVNumFmt** - CSV number separator
- **DataPath** - CSV or MetaStock data folder location
- **DataSource** - data source name
- **EndDate** - latest end date
- **EventListFile** - event list file path
- **ExcludeIf** - import filter formula (exclude symbol if true)
- **ExcludeList** - excluded symbol list or file path
- **Fundamentals** - Norgate current fundamental item list
- **KeepAdjusted** - keep all bar values split-adjusted in the data file
- **KeepRedundant** - keep redundant symbols for the same company
- **IncludeList** - included symbol list or file path
- **LogFile** - path\name of import log file to create
- **NoWeekends** - remove weekend bars
- **Padding** - type of padding (if any) to use for missing bars
- **SaveAs** - path\name of imported data (.RTD) file to save
- **StartDate** - earliest start date
- **SymInfoFile** - symbol information file path
- **Update** - Norgate data update request

## 17.11.4. Strategy Elements

---

All the elements of a trading strategy definition.

- **Allocation** - capital allocation formula
- **Ambiguity** - assumption to use when sequence of entry/target/stop is unknowable
- **BarSize** - strategy-specific timeframe
- **CalendarSym** - symbol to use as the market date list for this strategy
- **CashInOut** - daily deposit and/or withdrawal specification by formula
- **CashList** - CSV file with list of specific deposit and/or withdrawal amounts
- **Category** - position category formula
- **CloseSlip** - slippage formula for at-close market transactions
- **Commission** - commission formula
- **Compounded** - optionally overrides the default setting of *S.Compounded* for stats reporting
- **DebugEntry** -log output from a running test at *EntrySetup* evaluation time

- **DebugExit** - log output from a running test at *ExitRule* evaluation time
- **DebugTargetStop** - log output from running a test at *ExitLimit* / *ExitStop* evaluation time
- **EntryLimit** - entry limit price formula
- **EntryScore** - entry score formula
- **EntrySetup** - entry setup condition formula
- **EntrySkip** - entry skip condition formula
- **EntryStop** - entry stop price formula
- **EntryTime** - entry time constant
- **EntryTradeValue** - calculates a value to store in **T.ValueIn** item in the trade list record for this entry
- **ExitLimit** - exit limit price formula
- **ExitLimitQty** - share or contract quantity for partial limit-price exits
- **ExitLimitTime** - exit limit execution time
- **ExitQty** - share or contract quantity for partial at-market exits
- **ExitRule** - exit rule formula
- **ExitStop** - exit stop price formula
- **ExitStopQty** - share or contract quantity for partial stop-price exits
- **ExitStopTime** - exit stop execution time
- **ExitTime** - exit rule time constant
- **ExitTradeValue** - calculates a value to store in **T.ValueOut** item in the trade list record for this exit
- **LimitExtra** - limit price extra excursion formula
- **LimitSlip** - slippage formula for at-limit-price transactions
- **MarkToMarket** - whether strategy equity includes open-position mark-to-market value
- **MaxEntries** - maximum actual entries per day formula
- **MaxExposure** - maximum exposure percentage formula
- **MaxInvested** - maximum investment amount formula
- **MaxNewExp** - maximum new exposure per day formula
- **MaxNewInv** - maximum new investment per day formula
- **MaxNewPos** - maximum new positions per day formula
- **MaxPerTurn** - how many setups per selection turn a strategy can add
- **MaxPositions** - maximum open positions formula
- **MaxSameCat** - maximum same category open positions formula
- **MaxSameSym** - maximum same symbol open positions formula
- **MaxSetups** - maximum entry setups per day formula
- **OpenSlip** - slippage formula for at-open market transactions
- **OrderMktAsLmtPct** - allows generated market orders to optionally be converted to limit orders
- **OrderNote** - string to add to the text of each order and as the "note" value in CSV order lists
- **OrdersFile** - path/name of Alera orders (signals) file to generate

- **PriceRound** - order and trade price rounding interval
- **QtyFinal** - can be used to modify the quantity of an entry after top-down setup ranking has been done
- **QtyPrice** - which price (order vs. fill) to use when calculating Quantity, trade fraction, and exposure
- **QtyRound** - position size rounding interval
- **QtyType** - position size formula unit type
- **Quantity** - position size formula value
- **Reduce** - whether to reduce position size rather than skip the entry due to the *MaxExposure* and/or *MaxInvested* threshold
- **SetupScore** - entry setup score formula
- **SetupSkip** - setup skip condition formula
- **Side** - strategy side constant
- **Slippage** - general-purpose slippage formula
- **StopSlip** - slippage formula for at-stop-price transactions
- **StrategyScore** - value to use for this strategy when ranking all strategies to determine setup prioritization
- **TLAdjusted** - whether quantities and prices in an imported trade list are split-adjusted
- **TLDateFmt** - whether dates in an imported trade list are DMY or MDY
- **TLFields** - defines the column layout of an imported trade list CSV file
- **TLIgnoreRules** - whether strategy formulas are ignored in Test mode trade list playback
- **TLStratName** - strategy name within tradelist that maps to this script strategy
- **TLTimeShift** - number of hours to add or subtract to trade list entry dates/times
- **TradeList** - imported trade list file (CSV format)
- **Using** - strategy/benchmark/template to inherit from

## 17.11.5. Bar Data Values

---

Refer to the current bar in a scan or backtest. The *value[**offset**]* syntax can be used to refer to previous bars.

- **BarDate** - date of the current bar
- **BarNum** - number of this bar from start of data
- **BarsLeft** - number of bars remaining before the end of data
- **Close or C** - bar close price
- **Day** - day number
- **DayOfWeek** - day of week (Monday is 1)
- **DayOfYear** - day of year
- **Dividend** - dividend amount (\$/share)
- **EndOfMonth** - true if next bar will end in a different month than this one
- **EndOfWeek** - true if next bar will end in a different week than this one

- **Event** - user-defined value from Event List file
- **Extra** - bar extra value from CSV import
- **FunBar** - relative bar number within a multi-bar function calculation
- **High or H** - bar high price
- **InXXX** - index constituency flag (Norgate) as set during import
- **Low or L** - bar low price
- **Month** - month number
- **NextOpen** - next bar open price
- **Open or O** - bar open price
- **Range or R** - bar intraday range
- **Split** - bar split factor (unadjusted / adjusted)
- **TrueRange or TR** - bar range including prior close
- **Volume or V** - bar volume
- **Week** - week of year
- **Year** - year number

## 17.11.6. Indicator Functions

---

Calculate specific technical indicators using data relative to the current bar. The `indicator()[offset]` syntax can be used to calculate indicators relative to previous bars. Any parameter of an indicator, as well as the offset (if provided) can be a literal number, a single value, or a formula.

- **ADX** - Wilder's average directional index
- **ATR** - Wilder's average true range
- **BBBOT** - Bollinger band bottom
- **BBPCT** - Bollinger band percent (%B)
- **BBTOP** - Bollinger band top
- **BBTREND** - Bollinger band trend
- **BBWIDTH** - Bollinger band width
- **CCI** - commodity channel index
- **CRSI** - Connors RSI indicator
- **HVOL** - historical volatility
- **KBBOT** - Keltner band bottom
- **KBTOP** - Keltner band top
- **MACD** -  $MACD = EMA(C, len1) - EMA(C, len2)$
- **MACDH** -  $MACDH = MACD(len1, len2) - MACDS(len1, len2, len3)$
- **MACDS** -  $MACDS = MACD(len1, len2) - EMA(MACD(len1, len2), len3)$
- **MDI** - Wilder's negative directional index
- **OBV** - on balance volume
- **PDI** - Wilder's positive directional index

- **RRSI** - reverse RSI (price required for RSI to reach level)
- **RSI** - Wilder's relative strength index
- **SAR** - Wilder's parabolic stop and reverse
- **SS** - Grimes' Sigma Spike indicator
- **STOC** - stochastics

## 17.11.7. Multi-Bar Functions

---

Calculate any expression across multiple bars of any data series going back in time from the current bar. To start at an earlier bar, use the *function()[offset]* syntax. Any parameter of a function, as well as the offset (if provided) can be a literal number, a single value, or a formula.

- **AEMA** - adaptive exponential moving average
- **AESD** - adaptive exponential standard deviation
- **BBBotF** - Bollinger band bottom as a function
- **BBPctF** - Bollinger band percent (%B) as a function
- **BBTopF** - Bollinger band top as a function
- **BBTrendF** - Bollinger band trend as a function
- **BBWidthF** - Bollinger band width as a function
- **Correl** - correlation of two series
- **CountTrue** - count of all bars for which expression was true
- **DateBars** - count of bars since (or until) a specific date
- **EMA or XAvg** - exponential moving average
- **ESD** - exponential standard deviation
- **HMA or HAvg** - Hull moving average
- **Highest or HHV** - highest value
- **KAMA** - Kaufman adaptive moving average
- **Kurtosis** - sample kurtosis
- **LinReg** - linear regression
- **Lowest or LLV** - lowest value
- **MA or Avg** - simple moving average
- **Median** - median of values
- **Peak** - value of nth most recent peak followed by an n% drop (does not look ahead)
- **PeakBars** - count of bars since nth such Peak
- **PercentRank** - percent rank among values
- **PercentRankN** - value with nth percent rank
- **Product** - product of values
- **Rank** - numeric rank among values
- **RankN** - value with nth numeric rank
- **ROC or PctChg** - %gain/loss

- **RsiF** - RSI as a function
- **SarF** - SAR as a function
- **Sequence** - check for a sequence of conditions within a specific number of bars
- **SinceHigh** - bars since highest value
- **SinceLow** - bars since lowest value
- **SinceTrue** - bars since expression was true (0 if now, -1 if never)
- **Skewness** - sample skewness
- **Slope** - slope of linear regression
- **Spearman** - Spearman's Rank Correlation
- **StdDev** - standard deviation
- **StdErr** - standard error of predicted vs actual values in a regression
- **Sum** - sum of values
- **SumSince** - sum of values since condition was true
- **SumSQ** - sum of squared values
- **Trough** - value of nth most recent trough followed by an n% rise (does not look ahead)
- **TroughBars** - count of bars since nth such Trough
- **TrueInRow** - count of bars in a row for which expression was true
- **UntilTrue** - bars until expression will be true (0 if now, -1 if never)
- **WhenTrue** - value when expression was true (or will be true if count < 0)
- **WMA or WAvg** - weighted moving average
- **YInt** - y-intercept of linear regression

## 17.11.8. Cross-Sectional Functions

---

For use in the Data section to tag a data item as a cross-sectional function. Normally, a data formula is calculated by going back in time from the current bar for the given symbol. Cross-sectional items are instead calculated by going across all symbols on the given date. One use of cross-sectional data is to calculate ranking for a rotational strategy. Another is to generate market-breadth indicators.

- **#Avg** - cross-sectional average value
- **#ByCII** - groups values by corresponding industry index (?CII) for any cross-sectional function
- **#ByEcon** - groups values by economic sector (?EconSect) for any cross-sectional function
- **#ByGroup** - groups values by industry group (?IndGroup) for any cross-sectional function
- **#ByIndu** - groups values by industry (?Industry) for any cross-sectional function
- **#ByListNum** - groups values by Import IncludeList number
- **#ByMkt** - groups values by futures market (ES, CL, etc.) for any cross-sectional function
- **#BySect** - groups values by business sector (?Sector) for any cross-sectional function
- **#Count** - cross-sectional count of values
- **#DenseRank** - cross-sectional numeric rank of unique values
- **#Highest** - cross-sectional highest value
- **#Lowest** - cross-sectional lowest value

- **#Median** - cross-sectional median value
- **#OnePerDate** - calculate and store only one value per date (same value for all symbols)
- **#OnePerSym** - calculate and store only one value per symbol (same value for all dates)
- **#PercentRank** - cross-sectional percent rank
- **#Rank** - cross-sectional numeric rank
- **#StdDev** - cross-sectional standard deviation
- **#Sum** - cross-sectional sum of values

## 17.11.9. General-Purpose Functions

---

Perform general operations on single values. Any parameter of a function can be a literal number, a single value, or a formula.

- **Abs** - absolute value of a number
- **Bound** - limit the range of a value
- **Combined** - evaluate stats expression using combined strategy results
- **Cosine** - trigonometric cosine of a number of degrees
- **Cross** - value1[1] < value2[1] and value1 > value2
- **DataType** - causes value to be treated as a specific type (price vs. volume)
- **Date** - get the numeric date for a given year, month, day
- **Days** - number of calendar days between two dates
- **Exp** - exponential function ( $e^x$ )
- **Extern** - evaluate expression for a different stock/contract or strategy
- **IF** - conditional branch
- **IsNan** - true if value can't be evaluated, false if it can
- **Item** - refer to a Data or Library item by name
- **Log** - natural logarithm of a number
- **Max** - largest of a group of values
- **MaxN** - Nth largest of a group of values
- **Min** - smallest of a group of values
- **MinN** - Nth smallest of a group of values
- **NoNan** - evaluate an expression with no possibility of a nan (not a number) result
- **Random** - random number from min to max rounded to step
- **Round** - round value to nearest step
- **Select** - multi-part conditional branch
- **Sign** - sign of a number (1 or -1)
- **Sine** - trigonometric sine of a number of degrees
- **Sqr** - square root of a number
- **SymNum** - find the number of a given symbol, and/or allow dynamic external symbol usage
- **Tangent** - trigonometric tangent of a number of degrees

- **TargetPrice** - calculate the value of tomorrow's close that would cause an indicator to reach a specific level
- **Text** - literal text for scan output, with optional embedded numeric values
- **This** - anchored term in multi-bar function
- **Top** - top N digits of a number

## 17.11.10. String Functions

---

These functions operate on **string values**. Some of them return strings and others return numeric values.

- **Format** - embed numeric or string values within a string using formatting codes
- **Left** - get the left end of a string
- **Length** - get the number of characters in a string
- **Match** - see if a string matches a pattern
- **Mid** - extract part of a string
- **Replace** - replace text wherever it is found in a string
- **Right** - get the right end of a string
- **ToDate** - parse a string and return a date value
- **ToLower** - convert a string to lowercase
- **ToNum** - find a number within a string
- **ToUpper** - convert a string to uppercase

## 17.11.11. Stock/Contract Information

---

Information about the current stock in a test or scan.

- **?CII** - corresponding industry index symbol string
- **?Currency** - currency in which security trades
- **?Domicile** - country name string
- **?Exchange** - exchange name string
- **?EconSect** - economic sector name string
- **?EquityType** - security equity type (Common, Unit, etc.)
- **?IndGroup** - industry group name string
- **?Industry** - industry name string
- **?ListingType** - security listing type (Primary, ADR, etc.)
- **?Name** - security/company name string
- **?ReportingCurrency** - currency in which security reports earnings
- **?Sector** - business sector name string
- **?Symbol** - security symbol string

- **?Type** - security type string
- **InfoID** - Norgate Asset ID for this symbol
- **InfoExpiry** - futures contract expiration date
- **InfoFloat** - shares in circulation
- **InfoGICS** - Global Industry Classification Standard code
- **InfoMargin** - futures contract current margin requirement
- **InfoTRBC** - Thomson Reuters Business Classification code
- **F.xxx / F.xxx.Date** - Norgate current fundamental item value / date
- **FilterNum** - which filter was passed in a multi-filter scan
- **InList** - whether the stock was part of a specific **IncludeList** during import
- **ListNum** - number of first *IncludeList* the stock was part of during import
- **PointValue** - futures contract point value
- **InfoShares** - shares in existence
- **Symbol** - current symbol code
- **TickSize** - futures contract tick size

## 17.11.12. Current Position Information

---

Information about the current position in a test.

- **?Strategy** - name of current strategy (e.g. for *TestScan* output)
- **BarsHeld** - age of current position in bars
- **Category** - category specified for current position
- **EntryDate** - date on which current position was entered
- **FillFraction** - equity fraction of current position at time of fill
- **FillPrice** - entry fill price of current position
- **FillPriceAvg** - average fill price of current position if multiple positions
- **FillPriceMax** - highest fill price of current position if multiple positions
- **FillPriceMin** - lowest fill price of current position if multiple positions
- **FillQty** - shares or contracts in current position at time of entry
- **FillValue** - dollar value of current position at time of fill
- **IsOrder** - whether the current stock is a setup that was not skipped today
- **IsSetup** - whether the current is a setup today
- **OrderRank** - top-down order rank number for this position's entry across all strategies
- **OrderSum** - evaluates a formula for each non-skipped setup and returns the sum values
- **PrevExitLimit** - previous exit limit price for use in trailing target calculations
- **PrevExitStop** - previous exit stop price for use in trailing stop calculations
- **PositionSum** - evaluates a formula for each open position and returns the sum of values

- **SetupRank** - rank number by *SetupScore* for this position when it was entered
- **SetupSum** - evaluates a formula for each setup and returns the sum of values
- **Shares or Contracts** - number of shares or contracts in current position
- **StratNum** - ordinal number of current strategy in the script
- **TLValueIn** - the *ValueIn* value for a position that originated from trade list
- **TLValueOut** - the *ValueOut* value for a position that originated from trade list

### 17.11.13. Test Statistics Arrays

---

Used in the special-purpose scripts *results.rts* and *graphs.rts*, and can also be used in strategy formulas. Similar to a bar fields in that these represent a series of values, one per date.

- **S.Alloc** - current allocation amount
- **S.BPY** - strategy bars per year
- **S.CashInOut** - cumulative net cash in-out
- **S.Comms** - total commissions this period
- **S.Compounded** - strategy compounding flag
- **S.Date** - date of the current stat period
- **S.DDBars** - current drawdown duration
- **S.DDDlr** - current dollar drawdown
- **S.DDPct** - current percent drawdown
- **S.Dividends** - total dividends this period
- **S.Entries** - count of positions entered this period
- **S.EntryOrders** - number of entry orders that were placed this period
- **S.Equity** - current equity amount
- **S.Exits** - count of positions exited this period
- **S.Exposure** - net long-short dollars in overnight open positions
- **S.First** - period number in which first strategy trade entry occurred
- **S.Interest** - net interest this period
- **S.LossBars** - duration of losing trades this period
- **S.LossDlr** - dollar P&L of losing trades this period
- **S.Losses** - count of losing exits this period
- **S.LossPct** - percent P&L of losing trades this period based on position size
- **S.LossPctAlloc** - percent P&L of losing trades this period based on allocation
- **S.M2M** - net mark-to-market
- **S.MAE** - maximum adverse excursion (worst intraday drawdown)
- **S.MaxAlloc** - highest allocation amount
- **S.MaxDDBars** - longest drawdown duration
- **S.MaxDDdlr** - largest dollar drawdown
- **S.MaxDDPct** - largest percent drawdown

- **S.MaxEquity** - highest equity amount
- **S.MFE** - maximum favorable excursion (best intraday runup)
- **S.MinAlloc** - lowest allocation amount
- **S.MinEquity** - lowest equity amount
- **S.NetDlr** - dollar change in allocation value for this period
- **S.NetPct** - percent change in allocation value for this period
- **S.NetFx** - sum of net currency exchange rate change impact on trade profit or loss for this period
- **S.Number** - number of current stat period in a test
- **S.Positions** - count of overnight open positions
- **S.Setups** - total count of entry setups this period
- **S.Slips** - total slippage this period
- **S.StartEquity** - starting equity amount
- **S.Stops** - count of exits that were stops this period
- **S.Targets** - count of exits that were targets this period
- **S.TradeBars** - duration of all trades this period
- **S.TradeDlr** - dollar P&L of all trades this period
- **S.TradePct** - percent P&L of all trades this period based on position size
- **S.TradePcAlloc** - percent P&L of all trades this period based on allocation
- **S.TWEQ** - time-weighted equity value
- **S.Usage** - total long+short dollars in intraday open positions
- **S.WinBars** - duration of winning trades this period
- **S.WinDlr** - dollar P&L of winning trades this period
- **S.WinPct** - percent P&L of winning trades this period based on position size
- **S.WinPcAlloc** - percent P&L of winning trades this period based on allocation
- **S.Wins** - count of winning exits this period

## 17.11.14. Trade Record Values

---

These items provide detailed information about each specific trade in a test.

- **T.Bars** - trade duration (same-bar entry and exit is 0)
- **T.CommIn** - entry commision (\$)
- **T.CommOut** - exit commision (\$)
- **T.DateIn** - date of trade entry (numeric yyyyymmdd)
- **T.DateOut** - date of trade exit (yyyyymmdd)
- **T.Div** - net dividend received or paid (\$)
- **T.Fraction** - fraction of allocation at trade entry time that was used as the position size
- **T.FxIn** - currency exchange rate on trade entry date
- **T.FxOut** - currency exchange rate on trade exit date

- **T.Highest** - highest high during trade
- **T.Lowest** - lowest low during trade
- **T.NetFx** - currency exchange rate change impact on trade profit or loss
- **T.NetPct** - net trade profit after commission and dividend, expressed as a fraction of entry position size
- **T.Points** - net points gained or lost (\$/share)
- **T.PriceIn** - trade entry price
- **T.PriceOut** - trade exit price
- **T.Profit** - net trade profit after commission and dividend, expressed in dollars
- **T.PtVal** - point value of symbol of a trade
- **T.QtyIn** - shares or contracts bought or shorted
- **T.QtyOut** - shares or contracts sold or covered
- **T.Reason** - exit or skip reason code
- **T.Side** - side of a trade (1=long, -1=short)
- **T.SlipIn** - entry slippage (\$)
- **T.SlipOut** - exit slippage (\$)
- **T.SplitIn** - split factor (real / adj) at entry time
- **T.SplitOut** - split factor (real / adj) at exit time
- **T.Strat** - strategy number of a trade
- **T.TimeIn** - trade entry time-of-day code
- **T.TimeOut** - trade exit time-of-day code
- **T.ValueIn** - value calculated by **EntryTradeValue** when position was entered
- **T.ValueOut** - value calculated by **ExitTradeValue** when position was exited

These syntax elements are used most often in the special-purpose script **Trades.rts** or in a substitute **Trades Section**.

They can also be used in **Charts Section** and the **Trade Plot Options Dialog** formula.

As well, a **Strategy** can use these items to refer to **Past Trades** as part of its trading logic.

## 17.11.15. Trade Statistics Functions

---

These functions can be used in Results and/or Strategy formulas to drill down into the list of trades from the current test and calculate a few basic statistics about them.

Unlike the **Indicator** and **Multi-Bar** functions, the optional *Count* argument in these functions is a count of closed trades, not a count of bars or dates.

If the optional third argument *Symbol* is specified, *Count* applies to closed trades for that symbol.

Here are some example use cases:

```

▼TestData:
TSS1: TradeStatSum(T.NetPct, 10) // net %gain of the 10 most recently closed trades in any symbol
TSS2: TradeStatSum(T.NetPct, 10, $MSFT) // net %gain of the 10 most recently closed trades in MSFT
TSS3: TradeStatSum(T.NetPct, 10, Symbol) // net %gain of the 10 most recently closed trades in the current symbol
TSS4: TradeStatSum(if(Symbol = $MSFT, T.NetPct, 0), 100) // net %gain of any MSFT trades in the past 100 overall trades
TSS5: TradeStatSum(if(Symbol=This(Symbol), T.NetPct, 0), 100) // net %gain of any current-symbol trades in the past 100 overall trades

```

Most often the items that you'll want to reference within these formulas will be **Trade Record Values**.

When called from a **Strategy Element** formula, **Bar Data Values** can also be referenced.

When called from a **Results Section** formula, only the Trade Record Values are available.

If you need to reference a trade-specific data value such as an ATR indicator, you can calculate it during the test using **EntryTradeValue** or **ExitTradeValue** and then access it for the formulas below using **T.ValueIn** or **T.ValueOut**.

The following trade statistics functions are provided:

- **TradeStatAvg** - the average of trade record values for the most recent N trades or for all trades
- **TradeStatMax** - the largest of trade record values for the most recent N trades or for all trades
- **TradeStatMin** - the smallest of trade record values for the most recent N trades or for all trades
- **TradeStatStdDev** - the standard deviation of trade record values for the most recent N trades or for all trades
- **TradeStatSum** - the sum of trade record values for the most recent N trades or for all trades

## 17.12. Syntax Element Details

---

All the elements of the RealTest Script Language syntax are listed here in alphabetical order, each with a detailed description of its purpose and usage.

### 17.12.1. #Avg

---

#### Category

##### Cross-Sectional Functions

#### Description

For each date, evaluates a formula and calculates the average value for all symbols on that date.

#### Example

```
▼Data:  
// averate close of all stocks  
AvgPrice: #Avg C
```

### 17.12.2. #ByCII

---

#### Category

##### Cross-Sectional Functions

#### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of stocks that share the same corresponding industry index

#### Example

```
▼Data:  
StockRank: #Rank #ByCII if(InSPX, ROC(C, 30), -999)
```

Ranks \$SPX constituents separately by **Norgate** corresponding industry index.

## Notes

See also **?CII**, **CIIFamily**, and **CIILevel**.

See the **cii\_rotate.rts** sample script for a complete example.

## 17.12.3. #ByEcon

---

### Category

**Cross-Sectional Functions**

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of stocks that share the same economic sector name

### Example

```
▼Data:
EconRoc: #Avg #ByEcon ROC(C, 100)
```

Calculates the average 100-bar ROC of all stocks in this economic sector and stores it as "EconROC" for each stock.

### Notes

Requires data with **?EconSect** names in each stock record.

**Norgate** provides these automatically -- use **Classification** to specify which scheme to request at **Import** time.

For other data sources you would need to provide them via a **SymInfo** file.

## 17.12.4. #ByGroup

---

### Category

**Cross-Sectional Functions**

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of stocks that share the same industry group name

### Example

```
▼Data:
GroupRoc: #Avg #ByGroup ROC(C, 100)
```

Calculates the average 100-bar ROC for all stocks in this industry group and stores it as "GroupROC" for each stock.

### Notes

Requires data with **?IndGroup** names in each stock record.

**Norgate** provides these automatically -- use **Classification** to specify which scheme to request at **Import** time.

For other data sources you would need to provide them via a **SymInfo** file.

## 17.12.5. #ByIndu

---

### Category

## Cross-Sectional Functions

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of stocks that share the same industry name

### Example

```
▼Data:  
InduROC: #Avg #ByIndu ROC(C, 100)
```

Calculates the average 100-bar ROC for all stocks in this industry and stores it as "InduROC" for each stock.

### Notes

Requires data with **?Industry** names in each stock record.

**Norgate** provides these automatically -- use **Classification** to specify which scheme to request at **Import** time.

For other data sources you would need to provide them via a **SymInfo** file.

## 17.12.6. #ByListNum

---

### Category

#### Cross-Sectional Functions

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for the stocks from each separate **IncludeList**

### Example

```
▼Data:  
ListCount: #Sum #ByListNum 1
```

Counts the number of stocks in the same *IncludeList* as the current stock.

### Notes

The **ListNum** of a symbol is the number of the **first** *IncludeList* that it appears in.

The above example is therefore only useful when each *IncludeList* contains unique symbols.

The secondary cross-sectional ranking functions all work this way. For example a stock cannot belong to more than one sector or industry.

## 17.12.7. #ByMkt

---

### Category

#### Cross-Sectional Functions

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of individual futures contracts within the same market.

### Example

```
▼Data:  
MktRank: #Rank #ByMkt Volume
```

Rather than ranking all symbols in the data file by volume, each market-specific subset is ranked separately.

Having defined this data item, you could then simply refer to `MktRank` for the current symbol to see if it's the current highest-volume contract in its market.

See the example script `futures_volume_rank.rts` for a complete implementation of this.

Note that this mechanism only works with **Norgate** futures data, or with data that uses an identical naming convention.

See **Special Syntax for Individual Futures Contracts** for additional details.

## 17.12.8. #BySect

---

### Category

**Cross-Sectional Functions**

### Description

A secondary cross-sectional function, which requests that the primary function be calculated separately for each group of stocks that share the same business sector name

### Example

```
▼Data:
SectROC: #Avg #BySect ROC(C, 100)
```

Calculates the average 100-bar ROC of all stocks in this sector and stores it as "SectROC" for each stock.

### Notes

Requires data with **?Sector** names in each stock record.

**Norgate** provides these automatically -- use **Classification** to specify which scheme to request at **Import** time.

For other data sources you would need to provide them via a **SymInfo** file.

## 17.12.9. #Count

---

### Category

**Cross-Sectional Functions**

### Description

For each date, calculates the count of symbols for which a formula can be evaluated on that date.

### Example

```
▼Data:
// count of stocks with data
HasData: #Count C
```

### Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the count.

## 17.12.10. #DenseRank

---

### Category

**Cross-Sectional Functions**

## Description

For each date, evaluates a formula and then calculates the dense rank of each symbol's value among all symbols on that date. Lowest rank (1) means highest value. Identical values get the same rank number.

## Example

```
▼Data:
Factor:          ROC(C, 20)
SectScore:       #Avg #BySect Factor
DenseSectRank:  #DenseRank SectScore
RankInSect:     #Rank #BySect Factor
TopPick:        DenseSectRank <= 5 and RankInSect = 1
```

This first calculates a sector score as the average 20-day return of the symbols in that sector. All symbols in the same sector will have the same score. #DenseRank is then used to assign sector ranks to each symbol starting with 1 for the top sector, 2 for the next sector, etc. This makes it possible to determine e.g. the top stock from each of the top 5 sectors using a custom scoring factor.

## Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the ranking list (reducing the total count) and its rank value would be **nan** as well.

If the formula result is the same for two symbols then they both get the same rank number. Use **#Rank** if you want every symbol to have a unique rank number.

## 17.12.11. #Highest

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and calculates the highest (largest) value for all symbols on that date.

### Example

```
▼Data:
// highest-priced stock
HighPrice: #Highest C
```

## 17.12.12. #Lowest

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and calculates the lowest (smallest) value for all symbols on that date.

### Example

```
▼Data:
// lowest-priced stock
LowPrice: #Lowest C
```

## 17.12.13. #Median

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and calculates the median value for all symbols on that date.

### Example

```
▼ Data:
// median-priced stock
MedPrice: #Median C
```

### Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the count from which the median is derived.

## 17.12.14. #OnePerDate

---

### Category

#### Cross-Sectional Functions

### Description

Evaluate this **Data** or **TestData** formula only once per date and return this value for any stock that references the item

### Notes

This is most applicable to something like *Extern(\$SPY, C > MA(C,200))* where the result will be the same for all symbols.

Prior to release 2.0.26.1 RealTest would nevertheless evaluate that same formula redundantly for every date of every symbol, and waste memory storing all these redundant values.

For this specific example RealTest now automatically applies *#OnePerDate* even if not specified.

Add this tag explicitly at the start of any formula if to enforce this behavior.

See also **#OnePerSym**.

Add both *#OnePerDate* and *#OnePerSym* to calculate and store only a single value for all bars of all stocks (this is done automatically for constants and constant expressions).

## 17.12.15. #OnePerSym

---

### Category

#### Cross-Sectional Functions

### Description

Evaluate this **Data** or **TestData** formula only once per symbol and return this same value for all bars.

### Notes

This is most applicable to something like *Top(InfoTRBC, 4)* where the result will be the same for all symbols.

Prior to release 2.0.26.1 RealTest would nevertheless evaluate that same formula redundantly for every date of every symbol, and waste memory storing all these redundant values.

For this specific example RealTest now automatically applies `#OnePerSym` even if not specified.

Add this tag explicitly at the start of any formula if to enforce this behavior.

See also `#OnePerDate`.

Add both `#OnePerDate` and `#OnePerSym` to calculate and store only a single value for all bars of all stocks (this is done automatically for constants and constant expressions).

## 17.12.16. #PercentRank

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and then calculates the percent rank of each symbol's value among all symbols on that date.

### Notes

The largest value will have a percent rank of 100, the lowest value 0, and the others will be distributed evenly between those extremes.

### Example

```
▼ Data:
// percent rank of this stock's close
PriceRank: #PercentRank C
```

### Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the ranking list (reducing the total count) and its rank value would be **nan** as well.

## 17.12.17. #Rank

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and then calculates the rank of each symbol's value among all symbols on that date. Lowest rank (1) means highest value. Identical values get different rank numbers.

### Example

```
▼ Data:
// numeric rank of this stock's close
PriceRank: #Rank C
```

### Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the ranking list (reducing the total count) and its rank value would be **nan** as well.

If the formula result is the same for two symbols then the one that comes first alphabetically gets the lower rank number. Use `#DenseRank` if you want identical values to get the same rank number.

## 17.12.18. #StdDev

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and then calculates the standard deviation of values among all symbols on that date.

### Example

```
▼ Data:
// deviation of today's returns
PriceDev: #StdDev ROC(C, 1)
```

### Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the count that is used in the standard deviation calculation.

## 17.12.19. #Sum

---

### Category

#### Cross-Sectional Functions

### Description

For each date, evaluates a formula and calculates the sum of values for all symbols on that date.

### Example

```
▼ Data:
// total market volume today
TotalVol: #Sum V
```

## 17.12.20. ?CII

---

### Category

#### Stock/Contract Information

### Description

Symbol of the corresponding industry index (CII) of the current security

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.21. ?Currency

---

### Category

#### Stock/Contract Information

### Description

Name of the currency that the security trades in

#### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.22. ?Domicile

---

#### Category

**Stock/Contract Information**

#### Description

Name of the country of origin (domicile) of the security

#### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.23. ?EconSect

---

#### Category

**Stock/Contract Information**

#### Description

Name of the economic sector of the security (**TRBC** or **GICS** column 1)

#### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

This value is also used for two-level Data item ranking when **#Rank #ByEcon** is specified.

Use **Classification** to specify which scheme to request at **Import** time.

## 17.12.24. ?EquityType

---

#### Category

**Stock/Contract Information**

#### Description

Equity type of the security (Common/Ordinary, Unit, etc.)

#### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.25. ?Exchange

---

#### Category

**Stock/Contract Information**

#### Description

Name of the exchange that the security trades on

## Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.26. ?IndGroup

---

### Category

**Stock/Contract Information**

### Description

Name of the industry group of the security (**TRBC** or **GICS** column 3)

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

This value is also used for two-level Data item ranking when **#Rank #ByGroup** is specified.

Use **Classification** to specify which scheme to request at **Import** time.

## 17.12.27. ?Industry

---

### Category

**Stock/Contract Information**

### Description

Name of the industry of the security (**TRBC** or **GICS** column 4)

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

This value is also used for two-level Data item ranking when **#Rank #ByIndu** is specified.

Use **Classification** to specify which scheme to request at **Import** time.

## 17.12.28. ?ListingType

---

### Category

**Stock/Contract Information**

### Description

Listing type of the security (Primary, ADR, etc.)

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.29. ?LocalTime

---

### Category

General

## Description

Returns the current local date and time

## Notes

When used as a String the format is *yyyy-mm-dd hh:mm:ss*, e.g. "2023-12-21 14:40:05"

When used as a number the value is *yyyymmdd.hhmmss*, e.g. 20231221.144005

## 17.12.30. ?Name

---

### Category

**Stock/Contract Information**

### Description

Name of the security (company or security name)

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.31. ?ReportingCurrency

---

### Category

**Stock/Contract Information**

### Description

Name of the currency that the security reports earnings in

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

## 17.12.32. ?RunMode

---

### Category

General

### Description

Name of the mode in which the script is being run

### Notes

The return value is a **string** containing one of these run-mode names:

 Apply  Test  Optimize  Import  Scan  Orders

## 17.12.33. ?Sector

---

### Category

## Stock/Contract Information

### Description

Name of the business sector of the security (**TRBC** or **GICS** column 2)

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

This value is also used for two-level Data item ranking when **#Rank #BySect** is specified.

Use **Classification** to specify which scheme to request at **Import** time.

## 17.12.34. ?Strategy

---

### Category

#### Current Position Information

### Description

Name of the current strategy

### Notes

The return value is a **string** which can be used in **TestScan** or as input to any **String Function**.

## 17.12.35. ?Symbol

---

### Category

#### Stock/Contract Information

### Description

Symbol of the current security

### Notes

The return value is a **string** which can be used in the **Scan** or **Trades** section or as input to any **String Function**.

See also **Symbol**, which returns the current symbol as a **Symbol Constant**, and **SymNum**, which allows dynamic symbol lookup.

*?Symbol = "MSFT"* and *Symbol = \$MSFT* would both accomplish the same purpose, but it is slightly more efficient to use symbol constants.

Symbol constants also have the advantage of smart auto-complete when entering them.

## 17.12.36. ?Type

---

### Category

#### Stock/Contract Information

### Description

Type of the security

### Notes

When used in place of a formula, causes this text to be displayed in a column.

Can only be used in the **Scan** or **Trades** section.

## 17.12.37. Abs

---

### Category

#### General-Purpose Functions

### Description

Absolute Value of a number

### Syntax

Abs(value)

### Parameters

value - formula

## 17.12.38. AccountSize

---

### Category

#### Settings

### Description

Dollars in the simulated account at the beginning of each backtest

### Notes

If *AccountSize* is not specified in a script then the value from the **Settings Panel** will be used.

## 17.12.39. Adjustment

---

### Category

#### Import Specification

### Description

Norgate data adjustment setting

### Syntax

Adjustment: *choice*

### Choices

*TotalReturn* - all types of dividends are converted to splits

*CapitalSpecial* - corporate restructure events (mergers etc.) and special dividends are converted to splits

*Capital* - (default) no dividend types are converted to splits

*None* - adjustment information is not imported (not recommended)

### Notes

RealTest receives adjusted and unadjusted prices along with split and dividend histories from Norgate.

Imported data is then stored and used in unadjusted form so that each price reference on a given date reflects the actual price that was traded on that date.

When necessary for multi-bar indicator calculation or lookback comparisons, on-the-fly adjustment is performed.

See **Split Handling** for additional details about how this works.

General best practice is to use the default *Capital* in most cases.

Here are the trade-offs for each choice:

- *TotalReturn* converts all dividends to splits. This makes trades held across an ex-dividend date look like their share quantities were adjusted when they actually would not have been.
- *CapitalSpecial* does the above for special dividends including corporate restructuring events while correctly simulating payout of ordinary dividends.
- *Capital* only treats actual stock splits as splits while simulating payout as a dividend of every other adjustment event. This introduces an occasional large price gap (with corresponding simulated dividend payout) around corporate restructuring events.

## 17.12.40. ADX

---

### Category

#### Indicator Functions

### Description

Wilder's Average Directional Index

### Syntax

ADX(len)

### Parameters

len - lookback period

### Notes

Calculation uses the original Welles Wilder formula.

Wilder's exponential smoothing is equivalent to using  $2^{*len-1}$  in a regular exponential moving average.

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.41. AEMA

---

### Category

#### Multi-Bar Functions

### Description

Adaptive Exponential Moving Average

### Syntax

AEMA(expr, factor)

### Parameters

expr - data series formula

factor - weighting factor formula

## Notes

An AEMA is an **EMA** that supports a variable weighting factor. In other words, an EMA is an AEMA with a constant weight.

The *count* of an EMA becomes a weight factor using the formula  $factor = 2 / (count + 1)$ .

Conversely the *factor* of an AEMA can be converted to a EMA count using  $count = (2 - factor) / factor$ .

In both cases the average series is created by repeatedly calculating  $newAverage = oldAverage + factor * (newValue - oldAverage)$ .

The key difference between the functions is that while the EMA *count* is evaluated only once before the series of values is calculated, the AEMA *factor* is reevaluated for every bar of the series, thus making it *adaptive*.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.42. AESD

---

### Category

**Multi-Bar Functions**

### Description

Adaptive Exponential Standard Deviation

### Syntax

AESD(expr, factor)

### Parameters

expr - data series formula

factor - weighting factor formula

### Notes

This function calculates a standard deviation of values as an exponential series using a constant weighting factor represented as a bar count.

As with the **ESD** function, the weighting factor is used both to calculate an exponential mean of values and one of squared differences.

As with the **AEMA** function, the factor is equivalent to  $2 / (count + 1)$  of an ESD and is reevaluated for every bar of the calculation, thus making it *adaptive*.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.43. Allocation

---

### Category

**Strategy Elements**

### Description

Dollars allocated to this strategy

## Input

Any formula specifying a dollar amount

## Notes

In most scripts it is not necessary to specify *Allocation*.

It is generally not recommended to use *Allocation* to specify the fraction of the account that a strategy should use.

Instead apply the fraction via the **Quantity** formula when modeling combined compounding, or use **StartPercent** to specify an initial fraction when modeling strategies trading in separate accounts.

See **Asset Allocation and Position Sizing** for an example of each of these alternatives.

The default allocation is **Combined(S.Equity)** when no *StartPercent* is specified (all strategies in one account).

Strategies that specify *StartPercent* have a default allocation of **S.Equity** (each strategy in its own account).

Strategies that specify **Compounded: False** have a default allocation of **S.StartEquity** (maintain a constant dollar allocation).

The **S.Alloc** variable is updated daily by evaluating the *Allocation* formula if specified or using its default value otherwise.

*S.Alloc* is used implicitly in **Quantity** when **QtyType** is *Percent*.

*Quantity* should refer to *S.Alloc* rather than *S.Equity* when using either of the other quantity types (*Value* or *Shares*).

*Allocation* has **no effect** on the maximum investment level allowed for a strategy.

Investment level limits are specified using **MaxExposure** and/or **MaxInvested** and/or **MaxPositions**.

## 17.12.44. Ambiguity

---

### Category

**Strategy Elements**

### Description

Specifies what assumptions to make when there is price sequence ambiguity in a test

### Choices

*Default* - assume that if Close > Open then Low preceded High, or if Close < Open then High preceded Low (best guess)

*Stop* - always exit at the stop price if that price was touched (most pessimistic)

*Target* - always exit at the target (limit) price if that price was touched (most optimistic)

*Neither* - do not exit if exit price cannot be determined with zero ambiguity (most strict)

### Notes

This constant specifies what RealTest should do in a trade where more than one outcome could occur within the same bar and where a smaller timeframe bar would be required to know for sure which outcome happened first.

One example is a target and stop both being hit within a wide-range bar.

Another is a limit order entry and target exit within a single bar.

The "Default" (best guess) choice is used if Ambiguity is not specified, and in most situations this will

provide the most realistic results.

In order to see how often this setting is being applied, run your test with **TestOutput: Log** and then search the log file for the word "assuming".

## 17.12.45. Arg1-Arg9

---

### Category

**Library Section**

### Description

Special variables that can be used within a **Library** formula to access the arguments that were passed to it

### Notes

Library items can be referenced either with or without arguments. When referenced with arguments (so that the reference looks like a function call), the item formula can obtain the values passed in as arguments by using these numbered variables. *Arg1* is the first (leftmost) argument, *Arg2* the second one, and so on.

## 17.12.46. ATR

---

### Category

**Indicator Functions**

### Description

Wilder's Average True Range

### Syntax

ATR(len)

### Parameters

len - lookback period

### Notes

Calculation uses the original Welles Wilder formula.

Wilder's exponential smoothing is equivalent to using  $2^{*len-1}$  in a regular exponential moving average.

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.47. BarDate

---

### Category

**Bar Data Values**

### Description

Date of this bar

### Notes

Returns the current bar end date as a number in YYYYMMDD format.

If the bar has Weekly or Monthly **BarSize**, then this will be the closing date of the bar.

Use **BarStart** to get the opening date of the current bar.

Negative offsets, e.g. *BarDate[-5]*, can be legitimately used to obtain the date of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best results when future dates are required, a **HolidayList** should also be provided.

## 17.12.48. BarNum

---

### Category

**Bar Data Values**

### Description

Ordinal number of a bar within the data for that stock. The earliest bar is 1, next earliest is 2, etc.

## 17.12.49. BarsHeld

---

### Category

**Current Position Information**

### Description

The number of bars since this position was entered, not including the entry day

### Notes

For daily bars, it is easiest to think of this as *Nights Held*.

This will generally be used in the **ExitRule** formula to implement a **Time Stop**.

For example, if you entered on Monday and want to exit on Friday, your *ExitRule* formula would be *BarsHeld=4* (assuming there were no holidays that week). You are exiting the 4th trading day after your entry day.

If used as *Combined(BarsHeld)* when multiple positions are open in the same symbol (whether due to pyramiding or multiple strategies), the oldest entry date (largest value of *BarsHeld*) for that symbol will be returned.

See also **EntryDate**.

## 17.12.50. BarSize

---

### Category

**Settings** or **Strategy** element

### Description

Specifies the default bar size (periodicity) for scans and/or tests, or the strategy-specific bar size

### Notes

Valid values are *Daily*, *Weekly*, or *Monthly*.

If *BarSize* is not specified in a script then the value from the **Settings Panel** will be used.

See **Bar Sizes and Multiple Timeframes** for more information about how this works.

## 17.12.51. BarsLeft

---

### Category

#### Bar Data Values

### Description

The number of bars remaining before the end of data for this symbol

### Notes

BarsLeft is 0 for the last bar, 1 for the next-to-last bar, and so on.

This can be used in the **EntrySetup** formula to avoid entering positions in stocks that will soon be delisted. RealTest will automatically exit any position that remains open on the last data date (or last date of a test) at the close of that bar, so it is not necessary to use BarsLeft unless you want to be sure to exit before the last date.

BarsLeft can also be useful when modeling futures rollovers using individual contract data.

## 17.12.52. BarStart

---

### Category

#### Bar Data Values

### Description

Start Date of this bar

### Notes

Returns the current bar start date as a number in YYYYMMDD format.

Use **BarDate** to get the current bar end date.

If the bar has daily **BarSize**, then BarDate and BarStart are the same.

## 17.12.53. BBBOT

---

### Category

#### Indicator Functions

### Description

Bollinger band bottom

### Syntax

BBBOT(len, mult)

### Parameters

len - lookback period

mult - number of standard deviations

### Notes

This is equivalent to  $Avg(C, len) - (mult * StdDev(C, len))$ .

To calculate *BBBOT* for something other than *Close* use **BBBotF**.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.54. BBotF

---

### Category

#### Multi-Bar Functions

### Description

Bollinger band bottom as a function

### Syntax

```
BBotF(expr, len, mult)
```

### Parameters

expr - data series formula

len - lookback period

mult - number of standard deviations

### Notes

The **BBOT** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate the Bollinger band bottom of any series of values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.55. BBPCT

---

### Category

#### Indicator Functions

### Description

Bollinger band percent (%B)

### Syntax

```
BBPCT(len, mult)
```

### Parameters

len - lookback period

mult - number of standard deviations

### Notes

This indicator returns the position of Close relative to the Bollinger bands.

This is equivalent to  $(Close - BBOT(len, mult)) / (BBTOP(len, mult) - BBOT(len, mult))$ .

The return value is:

- >1 if close is above the top band
- 1 if close equals the top band
- 0.5 if close equals the moving average
- 0 if close equals the bottom band
- <0 if close is below the bottom band

To calculate *BBPCT* for something other than *Close* use **BBPctF**.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.56. BBPctF

---

### Category

#### Multi-Bar Functions

### Description

Bollinger band percent (%B) as a function

### Syntax

BBPctF(expr, len, mult)

### Parameters

expr - data series formula

len - lookback period

mult - number of standard deviations

### Notes

The **BBPCT** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate the %B of any series of values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.57. BBTOP

---

### Category

#### Indicator Functions

### Description

Bollinger band top

### Syntax

BBTOP(len, mult)

### Parameters

len - lookback period

mult - number of standard deviations

### Notes

This is equivalent to  $Avg(C, len) + (mult * StdDev(C, len))$ .

To calculate *BBTOP* for something other than *Close* use **BBTopF**.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.58. BBTopF

---

### Category

#### Multi-Bar Functions

### Description

Bollinger band top as a function

### Syntax

BBTopF(expr, len, mult)

### Parameters

expr - data series formula

len - lookback period

mult - number of standard deviations

### Notes

The **BBTOP** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate the Bollinger band top of any series of values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.59. BBTREND

---

### Category

#### Indicator Functions

### Description

Bollinger band trend

### Syntax

BBTREND(len1, len2, mult)

### Parameters

len1 - first lookback period

len2 - second lookback period

mult - number of standard deviations

### Notes

The BBTREND indicator is calculated as follows:

$$\text{lower} = \text{ABS}(\text{BBBOT}(\text{len1}, \text{mult}) - \text{BBBOT}(\text{len2}, \text{mult}))$$
$$\text{upper} = \text{ABS}(\text{BBTOP}(\text{len1}, \text{mult}) - \text{BBTOP}(\text{len2}, \text{mult}))$$
$$\text{BBTREND} = (\text{lower} - \text{upper}) / \text{Avg}(\text{C}, \text{len1})$$

This indicator is intended to signal both strength and direction of trend.

To calculate *BBTREND* for something other than *Close* use **BBTrendF**.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.60. BBTrendF

---

## Category

### Multi-Bar Functions

## Description

Bollinger band trend as a function

## Syntax

```
BBTrendF(expr, len1, len2, mult)
```

## Parameters

expr - data series formula

len1 - first lookback period

len2 - second lookback period

mult - number of standard deviations

## Notes

The **BBTREND** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate the Bollinger band trend of any series of values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.61. BBWIDTH

---

## Category

### Indicator Functions

## Description

Bollinger band width

## Syntax

```
BBWIDTH(len, mult)
```

## Parameters

len - lookback period

mult - number of standard deviations

## Notes

This indicates the width of the Bollinger bands relative to the average price that they surround.

This is equivalent to  $100 * (BBTOP(len, mult) - BBBOT(len, mult)) / Avg(C, len)$ .

To calculate *BBWIDTH* for something other than *Close* use **BBWidthF**.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.62. BBWidthF

---

## Category

### Multi-Bar Functions

## Description

Bollinger band width as a function

### Syntax

BBWidthF(expr, len, mult)

### Parameters

expr - data series formula

len - lookback period

mult - number of standard deviations

### Notes

The **BBWIDTH** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate the Bollinger band width of any series of values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.63. Benchmark

---

### Category

**Script Sections**

### Description

Defines a benchmark strategy

### Notes

See **Special Strategy Types** for details.

## 17.12.64. Bound

---

### Category

**General-Purpose Functions**

### Description

Force a value to be between a lower and upper limit

### Syntax

Bound(value, low, high)

### Parameters

value - formula specifying a value to bound

low - formula specifying a lower boundary value

high - formula specifying an upper boundary value

## 17.12.65. CalendarSym

---

### Category

**Strategy Elements**

## Description

Symbol to use as the market date list for this strategy

## Input

Any symbol in the current data file (does not require an extra \$)

## Notes

*CalendarSym* is only needed when a multi-strategy script includes strategies for different countries or for markets with different trading calendars.

An global date list is needed when evaluating formulas that don't have a specific symbol context, e.g. *MaxPositions*, *MaxExposure*, etc.

The global date list includes every date that has a bar for any stock in the data file.

Specifying *CalendarSym* replaces the default date list with a specific symbol's date list for this strategy only.

An example of the problem that this optional element solves:

- it is Tuesday morning and Monday was a US holiday
- your data file includes the symbol AUDUSD which has a bar for Monday
- the global date list therefore includes Monday's date
- your strategy uses a *MaxPositions* formula that varies depending on a US-index-based trend filter
- while trying to evaluate *MaxPositions* for Tuesday's trades, it sets the context to Monday's date, based on the global date list
- there is no Monday bar for US indices, so references to them return nan
- *MaxPositions* is therefore 0

Adding e.g. *CalendarSym: \$SPX* to the strategy would resolve this issue.

(Importing with *Padding: AllMarketDays* would also resolve it but there may be reasons you don't want to do that.)

## 17.12.66. CashInOut

---

### CashInOut

#### Strategy Elements

## Description

Specifies deposits and/or withdrawals to/from the current equity of a strategy by formula

## Input

Formula expression returning an amount in dollars

## Notes

This formula is evaluated once per day at the beginning of the day, before any trading signals are processed.

The resulting amount is added to the equity of the strategy.

The formula should return a positive value for deposits or a negative value for withdrawals, or 0 on dates when neither occurs.

The simplest way to use this formula in a multi-strategy system is to give it its own strategy, as in the examples below.

To provide a list of specific dates and amounts, use **CashList** rather than *CashInOut*.

## Examples

Add \$1,000 every month at the start of the month:

```
▽Strategy: cash  
CashInOut: if(Month<>Month[1], 1000, 0)
```

Withdraw 40% of last year's net profit at the start of each year:

```
▽Strategy: cash  
CashInOut: if(Year[-1]>Year, -0.4 * max(0, Combined(Sum(S.NetDlr, S.BPY))), 0)
```

## 17.12.67. CashIntPct

---

### Category

#### Settings

### Description

Interest rate received for positive daily excess cash

### Notes

If the settings also provide a **RiskFreeRateSym** then *CashIntPct* is **added** to the current daily interest rate as determined by today's value of the risk-free rate series. In this case, *CashIntPct* would typically be negative, e.g. -0.5 if your broker pays 0.5% less than the fed funds rate (with floor of zero).

If *RiskFreeRateSym* is provided but *CashIntPct* is not provided or is 0 then no interest is received. You must specify a non-zero value of *CashIntPct* to include interest on excess cash in your backtests.

If *CashIntPct* is provided when there is no *RiskFreeRateSym* then *CashIntPct* is simply a fixed annual interest rate.

Daily net interest received or paid is added to combined **S.Equity** and therefore becomes part of the total return of the backtest.

The stats series **S.Interest** can be used to graph or reference the cumulative net interest received or paid in the account.

See also **MarginIntPct** which specifies the rate charged for negative excess cash (margin loan).

## 17.12.68. CashList

---

### CashList

#### Strategy Elements

### Description

Specifies a CSV file that lists deposits and/or withdrawals to/from the current equity of a strategy

### Input

Path to a CSV file with dates in the first column and amounts in the second column

### Notes

The file can have any number of rows, and can have multiple rows for the same date.

The header row is optional.

Use positive amounts for deposits and negative amounts for withdrawals.

Amounts for each date in a test are applied at the beginning of that day, before any trading signals are processed.

If dates are weekend or holiday, they are applied to the next trading day.

To specify deposits and withdrawals by formula, use **CashInOut** rather than *CashList*.

### Example

*CashList* is most often used with strategies that use **TradeList** to play back live trades:

```
▼ Strategy: live_strat_1
  TradeList: c:\live_trading\trades.csv
▼ Strategy: live_strat_2
  TradeList: c:\live_trading\trades.csv
▼ Strategy: cash
  CashList: c:\live_trading\in-out.csv
```

The CSV file would look like this (additional columns will be ignored):

Date	Amount
1/15/2022	-5000
2/14/2022	-5000
etc.	

## 17.12.69. Category (reference)

---

### Category

#### Current Position Information

### Description

The value of the **Category** strategy element formula for this position

### Notes

This property can be referenced in any strategy element formula. It is most likely useful in **EntrySetup**, **Quantity**, or **ExitRule**.

## 17.12.70. Category (definition)

---

### Category

#### Strategy Elements

### Description

Assigns a category value to a position at entry time

### Input

Formula expression

### Notes

This formula is evaluated before the **EntrySetup** for each stock each day, so its value can be accessed in any other entry-related formula using the **Category** property.

A typical usage, along with the **MaxSameCat** element, is to limit the number of simultaneous positions from the same sector or industry.

When using Norgate data, either the **TRBC** or **GICS** classification value for the current symbol can

be referenced in the Category formula. Using TRBC along with the **Top** function is an especially convenient way to specify a category for this purpose. For example, *Top(TRBC,2)* is the economic sector, *Top(TRBC,4)* is the business sector, *Top(TRBC,6)* is the industry group, and *Top(TRBC,8)* is the specific industry.

## 17.12.71. CCI

---

### Category

#### Indicator Functions

### Description

Commodity Channel Index

### Syntax

CCI(len)

### Parameters

len - lookback period

### Notes

For the best description of this indicator, see its [StockCharts ChartSchool](#) page.

## 17.12.72. Charts

---

### Category

#### Script Sections

### Description

Chart indicator definitions

### Notes

See [Charts Section](#) and [Candlestick/Bar Charts](#).

## 17.12.73. CIIFamily

---

### Category

#### Import Specification

### Description

**Norgate** corresponding industry index family

### Syntax

CIIFamily: *choice*

### Choices

*\$SPX* - S&P 500 industry indices

*\$SP1500* - S&P 1500 industry indices

*\$XJO* -ASX 200 industry indices

*\$XKO* - ASX 300 industry indices

## Notes

Used in conjunction with **CIILevel** to add corresponding index references to each import stock symbol.

Causes the relevant index symbols to be automatically added to the import as well.

Automatically-added index symbols are placed in virtual **ListNum = 99**.

The specific industry index for the current symbol can be referenced using **Extern(&99, expression)**.

## 17.12.74. CIILevel

---

### Category

**Import Specification**

### Description

**Norgate** corresponding industry index level

### Syntax

CIILevel: *choice*

### Choices

*EconomicSector* - use economic sector indices

*IndustryGroup* - use industry group indices

*SpecificIndustry* - use specific industry indices

*SubIndustry* - use sub-industry indices

### Notes

Used in conjunction with **CIIFamily** to add corresponding index references to each import stock symbol.

Causes the relevant index symbols to be automatically added to the import as well.

Automatically-added index symbols are placed in virtual **ListNum = 99**.

The specific industry index for the current symbol can be referenced using **Extern(&99, expression)**.

## 17.12.75. Classification

---

### Category

**Import Specification**

### Description

Name of the classification scheme to use when querying Norgate for each stock's sector and industry names

### Choices

*TRBC* - use **The Refinitive Business Classification**

*GICS* - use the **Global Industry Classification Standard**

### Notes

Four classification names are imported for each stock: **?EconSect**, **?Sector**, **?IndGroup**, and **?Industry**.

In each of the two schemes, the first four column values (see above links) are assigned to those

names respectively.

As well as for display purposes, these names are used when calculating two-level rankings e.g. `#Rank #ByIndGroup factor`.

## 17.12.76. Close or C

---

### Category

#### Bar Data Values

### Description

Current bar closing price

### Notes

Either *Close* or *C* can be used as the name of this value.

## 17.12.77. CloseSlip

---

### Category

#### Strategy Elements

### Description

Slippage amount, in points (dollars per share or contract), for each transaction that simulates a market order filling at the close

### Input

Any formula specifying dollars per share or contract (points)

### Notes

Defines the amount of slippage to apply to each market-at-close transaction, in price points.

*CloseSlip* is applied to any transaction that logically occurs at the close and not at a specified limit or stop price.

If *CloseSlip* is not specified then **Slippage** is applied instead.

## 17.12.78. Combined (function)

---

### Category

#### General-Purpose Functions

### Description

evaluate stats expression using combined strategy results

### Syntax

Combined(expression)

### Parameters

expression - formula

### Notes

This function is intended for use with strategy-specific syntax elements such as **S.Equity** or **Shares**.

## 17.12.79. Combined (section)

---

### Category

#### Script Sections

### Description

Allows definition of all-strategy combined constraints such as **MaxExposure**, **MaxInvested**, **MaxPositions**, etc.

### Notes.

Combined capacity constraints are ignored if **Legacy Mode** is used.

See **Special Strategy Types** for details.

## 17.12.80. Commission

---

### Category

#### Strategy Elements

### Description

Commission amount, in dollars, for each trade

### Input

Formula specifying a dollar amount

### Notes

If your broker charges no commissions, omit this formula or set it to 0.

If you pay a flat fee for every trade, simply specify that number.

Commission is calculated and charged separately for entry and exit transactions, so round-trip commission is twice the value of this formula.

If **FillPrice** is used in the *Commission* formula, it will automatically retrieve the entry price for the entry commission and the exit price for the exit commission.

The formula for the standard US commission at Interactive Brokers is:  $\text{Min}(0.01 * \text{FillValue}, \text{Max}(0.005 * \text{Shares}, 1))$ .

IB's Canadian commission formula is:  $\text{Min}(0.005 * \text{FillValue}, \text{Max}(0.01 * \text{Shares}, 1))$  -- yes that's correct, the 0.01 and 0.005 are reversed vs. US.

The Australian commission formula for IB is:  $1.1 * \text{Max}(6.00, 0.0008 * \text{FillValue})$  -- the 1.1x is the 10% GST and the remainder is the IB commission.

For US IB commissions, you can omit the "Min(0.01 \* FillValue)" part unless you intend to trade penny stocks. By definition a stock would need to be priced below \$0.50/share for 1% of its value to exceed \$0.005/share.

## 17.12.81. Compounded

---

### Category

#### Strategy Elements

### Description

Optionally overrides the automatically-determined **S.Compounded** flag for a strategy

## Choices

*True* - force the strategy to report stats as if compounded

*False* - force the strategy to report stats as if non-compounded

*(unspecified)* - report stats as compounded if equity is compounded or as non-compounded if equity is not compounded (default)

## Notes

This setting mainly controls how percent-based stats are reported.

For tests where *Compounded* was *True*, the "ROR" stat is *compound annual return* (CAR) and the denominator used to calculate percent drawdown is the peak equity value.

For tests where *Compounded* was *False*, the "ROR" stat is *average annual return* (AAR) and the denominator used to calculate percent drawdown is the starting (constant) equity value.

If **Allocation** was not specified then it will default to **Combined(S.Equity)** when *Compounded* is *True* or to **S.StartEquity** when *Compounded* is *False*.

This in turn impacts **Quantity** (position size) calculations as the strategy runs.

To deliberately show compounded stats for a non-compounded test, explicitly specify *Allocation: S.Equity* or *Allocation: Combined(S.Equity)* while using **QtyType** of either *Shares* or *Value* and a *Quantity* formula that refers to neither **S.Alloc** nor **S.Equity**.

To deliberately show non-compounded stats for compounded test, explicitly specify *Allocation: S.StartEquity* while using *QtyType* of either *Shares* or *Value* and a *Quantity* formula that refers to *S.Equity*.

See also **Compounding**.

## 17.12.82. Constituency

---

### Category

**Import Specification**

### Description

Norgate index constituency list

### Syntax

Constituency: \$SPX, \$DJI // etc.

### Notes

Norgate supports historical index constituency series for many different indexes.

The following links to their website will provide the latest details:

- **US Historical Index Constituents**
- **AU Historical Index Constituents**
- **CA Historical Index Constituents**

These details can also be found in the **constituency.csv** file that comes with RealTest and is automatically loaded each time the program starts.

Here is a subset of that file:

	A	B	C	D
1	1	\$MEL	InMEL	major exchange listed stock
2	2	\$XAO	InXAO	All Ordinaries
3	3	\$SPASX101-500	InSPASX101_500	All Ordinaries excl S&P ASX 100
4	4	\$SPASX201-500	InSPASX201_500	All Ordinaries excl S&P ASX 200
5	5	\$SPASX301-500	InSPASX301_500	All Ordinaries excl S&P ASX 300
6	6	\$DJI	InDJI	Dow Jones Industrial Average
7	7	\$NDX	InNDX	Nasdaq 100
8	8	\$N1-200	InN1_200	Nasdaq 100 + Next Generation 100 Superset
9	9	\$N1-150	InN1_150	Nasdaq 100 + Q-50 Superset
10	10	\$NDXT	InNDXT	Nasdaq 100 Technology Sector
11	11	\$NBI	InNBI	Nasdaq Biotechnology
12	12	\$QNET	InQNET	Nasdaq Internet
13	13	\$NGX	InNGX	Nasdaq Next Generation 100
14	14	\$NXTQ	InNXTQ	Nasdaq Q-50
15	15	\$RUI	InRUI	Russell 1000
16	16	\$R1-4000	InR1_4000	Russell 1000 + 2000 + Micro Cap Superset
17	17	\$RUT	InRUT	Russell 2000
18	18	\$R2001-4000	InR2001_4000	Russell 2000 + Micro Cap Superset

Column A contains the index number used by RealTest internally to identify each constituency time series.

Column B contains the symbol to use in your *Constituency* statement if you want to include that constituency time series in your imported data.

Column C is the **InXXX** variable to reference to use in your script formulas to find out whether the current stock was a member of that index on the current date.

Column D is the name of the standard **Norgate** watchlist containing current members of that index.

When your **Import** definition includes a *Current & Past* version of any of these standard watchlists, RealTest automatically adds the *Constituency* series for each such index to your imported data IF your script does NOT include a *Constituency* statement.

Therefore you may rarely need to specify *Constituency* in an *Import* definition -- do so only when importing your own custom watchlists but still need to use *InXXX* in some of your formulas. An *Constituency* specification, if provided, must include all the indices for which *InXXX* is needed.

The first row above is a special case. It refers to Norgate's **US Major Exchange Listed** time series. Add *Constituency: \$MEL* to your import and refer to *InMEL* in your formulas to ensure that a stock was listed on a major exchange (not over-the-counter traded) on that date. (In practice this makes very little difference if you already use a reasonable liquidity filter.)

Similarly (but not shown above) add *Constituency: \$SPAC* to your import and refer to *InSPAC* in your formulas to determine whether and when a stock was trading as a "Special Purpose Acquisition Company".

See **The Event List File** for details about how *Constituency* data is stored in an RTD file and how to provide your own such data when importing from CSV or other sources.

## 17.12.83. Correl

---

### Category

#### Multi-Bar Functions

### Description

Correlation of two series

### Syntax

Correl(expr1, expr2, count)

### Parameters

expr1 - data series formula

expr2 - data series formula

count - lookback period

### Notes

*Correl(C, Extern(\$SPY,C), 100)* would be a simple way to calculate the rolling 100-day correlation of a stock to SPY.

## 17.12.84. Cosine

---

### Category

**General-Purpose Functions**

### Description

Trigonometric cosine of a number of degrees

### Syntax

Cosine(value)

### Parameters

value - formula

### Notes

The parameter value is assumed to be degrees (0-360).

To convert radians to degrees, multiply by 57.2957795131 (180/π).

## 17.12.85. Cross

---

### Category

**General-Purpose Functions**

### Description

Compares two values for two bars to see if their ranking has changed

### Syntax

Cross(value1, value2)

### Parameters

value1 - formula related to bar data

value2 - formula related to bar data

### Notes

This function is a shortcut for the expression  $value1[1] < value2[1]$  and  $value1 > value2$ .

It is most commonly used to compare two moving averages, or price to a moving average, e.g. *Cross(MA(C,5), MA(C,20))* would return 1 (true) if the 5-day average was below the 20-day average yesterday and is above it today.

Logically, *Cross(A,B)* means "A has crossed above B". To test for "A has crossed below B", simply use *Cross(B,A)*.

## 17.12.86. CountTrue

---

### Category

#### Multi-Bar Functions

### Description

Count of bars for which a condition was true (non-zero)

### Syntax

```
CountTrue(condition, count {0})
```

### Parameters

condition - data series formula

count - lookback period (optional)

### Notes

*Condition* will always be evaluated for *count* bars (or all bars if omitted).

For each bar, *condition* is evaluated as if that bar were the current bar, i.e. without knowledge of *future* splits relative to that bar.

If *condition* was never true for any bar, the return value is 0.

Unless you really care about the specific count, it is more efficient to use **SinceTrue** to test whether a condition has ever been true.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without a count argument.

## 17.12.87. CRSI

---

### Category

#### Indicator Functions

### Description

Connors RSI Indicator

### Syntax

```
CRSI(lenRocRank, lenRsiPrice, lenRsiStreak)
```

### Parameters

lenRocRank - number of bars to use in the PercentRank(ROC(C,1), len) portion of the calculation

lenRsiPrice - length parameter for the price RSI portion of the calculation

lenRsiStreak - length parameter for the "streak" RSI portion of the calculation

### Notes

This calculates the equivalent of the following formula:

```
(PercentRank(ROC(C,1), lenRocRank) + RSI(lenRsiPrice) + RSIF(TrueInRow(C > C[1]) - TrueInRow(C < C[1]), lenRsiStreak)) / 3
```

*CRSI* was added as a built-in function to improve calculation speed and enable "reverse CRSI" to be calculated by passing it to the **TargetPrice** function.

## 17.12.88. CSVDateFmt

---

### Category

#### Import Specification

### Description

Specifies whether the dates in a **CSV Import** are in M/D/Y vs. D/M/Y format

### Choices

*DMY* - dates are D/M/Y

*MDY* - dates are M/D/Y

### Notes

The default if your **Import** definition does not include *CSVDateFormat* is to use the *Date Display Format* setting from the **Program Options Dialog**.

This element lets you use CSV data files with the opposite of your standard date format more easily.

**Other date formats** are supported which are not ambiguous and therefore do not require *CSVDateFmt* to be specified.

## 17.12.89. CSVDelim

---

### Category

#### Import Specification

### Description

Specifies the column delimiter in a **CSV Import** file

### Choices

*Comma*

*Semicolon*

*Tab*

### Notes

The default column delimiter is *Comma* when **CSVNumFmt** is *Point* or *Semicolon* when **CSVNumFmt** is *Comma*.

## 17.12.90. CSVFields

---

### Category

#### Import Specification

### Description

CSV field order (comma-separated list)

### Choices

*Date* - this field is the bar date

*Time* - this field is the bar time

*Open* - this field is the open price

*High* - this field is the high price

*Low* - this field is the low price

*Close* - this field is the close price

*Volume* - this field is the volume

*AdjClose* - this field is the adjusted close price

*RealClose* - this field is the unadjusted (as-traded) close price

*Dividend* - this field is the dividend amount (should be 0 except on ex-dividend dates)

*Extra* - this field is the value that can be referenced as **Extra**

## Notes

CSV field order must be specified for **CSV Import** to work. If CSV data files include header rows with column labels, these are simply ignored.

Here is a partial example of a CSV import specification showing how this looks:

```
▼ Import:
  DataSource: CSV
  CSVFields: date,open,high,low,close,volume,realclose,dividend
```

Note that CSV field names (like all names in RealTest) are not case-sensitive.

To ignore a column in a CSV file, add an extra comma to the field order list, e.g. "date,,,open,high,low,close,,volume".

To ignore the first one or more columns, add extra comma(s) at the start of the list, e.g. ",,date,close".

## 17.12.91. CSVFile

---

### Category

#### Import Specification

### Description

CSV file to be used for single-file CSV data import

### Input

The full path to a file that contains CSV data for one or more symbols.

### Notes

The file must include a *Symbol* column, and **CSVFields** must specify its location.

For multi-file CSV import (one file per symbol), use **DataPath** instead of *CSVFile*.

See **CSV Import** for additional details.

## 17.12.92. CSVNumFmt

---

### Category

#### Import Specification

### Description

Specifies whether numbers (prices) in a **CSV Import** use a point or a comma as their decimal separator

### Choices

*Point* - point (period) is used (default)

*Comma* - comma is used

## Notes

By default (and by definition) comma is the field separator in a CSV file, and point is the decimal separator.

This optional item enables European-format "CSV" files to be imported by RealTest. When you add "CSVNumFmt: Comma" to your CSV import definition, semicolon becomes the default **CSVDelim** and comma is used as the decimal separator.

Such files sometimes use point as the date field separator, and RealTest also supports this format automatically. See also **CSVDateFmt** for D/M/Y date field order specification.

## 17.12.93. Currency

---

### Category

**Settings**

### Description

The base currency of the model account in a backtest

### Value

A three-letter currency abbreviation such as AUD, CAD, USD, JPY etc. (do not include quotation marks)

### Notes

Only specify a base currency when you specifically want to test multi-currency strategies, e.g. trading US stocks in a non-US account.

For this setting to work, the active data file must include the appropriate exchange rate series, e.g. AUDUSD, and the stock metadata must specify the currency for each symbol.

See **Testing Multi-Currency Strategies** for details on how this works.

## 17.12.94. Data

---

### Category

**Script Sections**

### Description

Named formulas calculated once and stored in memory arrays before tests are run

### Notes

See **Data Section** for a more detailed description.

## 17.12.95. DataFile

---

### Category

**Settings**

### Description

Path to the .RTD data file to load before running a scan or test

### Notes

If *DataFile* is not specified in a script then the path from the **Settings Panel** will be used.

## 17.12.96. DataPath

---

### Category

**Import Specification**

### Description

CSV or MetaStock file location

### Input

The full path to a directory (folder) that contains a collection of CSV or MetaStock data files.

### Notes

See **CSV Import** or **MetaStock Import** for details.

## 17.12.97. DataSource

---

### Category

**Import Specification**

### Description

Name of the data source to use for a data import

### Choices

*Norgate* - import from **Norgate NDU** (stocks, futures, indexes, indicators)

*Yahoo* - import from **finance.yahoo.com** (stocks)

*Tiingo* - import from **api.tiingo.com** (stocks)

*TiingoCrypto* - import end-of-day Crypto currency prices from Tiingo

*Metastock* - import stock data from a **local Metastock** database

*CSV* - import any type of data from your own **local CSV files**

## 17.12.98. DataType

---

### Category

**General-Purpose Functions**

### Description

Causes a value to be treated as a specific type of data

### Syntax

DataType(expression, type)

### Parameters

expression - any formula or value

type - what type to assign to the result: 0 = none, 1 = price, 2 = volume

### Notes

This function is rarely needed because RealTest automatically determines the type of most calculated values correctly.

The data type of a value is only important when the value is used in multi-bar formulas or indicators that cross a split date.

Only use this function if you have observed incorrect split handling in, for example, a complex Data Section formula.

## 17.12.99. Date

---

### Category

#### General-Purpose Functions

### Description

Returns a numeric date value for a given year, month and day

### Syntax

Date(year, month, day)

### Parameters

year - year number

month - month number

day - day number

### Notes

The return value is the date in YYYYMMDD format, which can be used for comparison with **BarDate** or as input to **DateBars**.

## 17.12.100. DateBars

---

### Category

#### Multi-Bar Functions

### Description

Number of bars since (or until) a specific date

### Syntax

DateBars(date)

### Parameters

date - a numeric date

### Notes

This function can be used to return a bar count or offset from the current bar to a specific date in the past or future.

The *date* parameter can be any formula returning a numeric date value.

The **Date** function is a convenient way to generate a numeric date.

If there is no bar with the specified date, the first bar with a date more recent than the specified date is used.

### Examples

*PctChg(C, DateBars(Date(2016,2,1)))* returns the percent gain or loss since February 2, 2016.

*C[DateBars(Date(Year,1,1))]* returns the closing price for the first trading date of the current year.

## 17.12.101. DateDay

---

### Category

**General-Purpose Functions**

### Description

Extract the day number from a date

### Syntax

DateDay(date)

### Parameters

date - an integer date value e.g. 20230816, S.Date, T.DateIn, etc.

## 17.12.102. DateMonth

---

### Category

**General-Purpose Functions**

### Description

Extract the month number from a date

### Syntax

DateMonth(date)

### Parameters

date - an integer date value e.g. 20230816, S.Date, T.DateIn, etc.

## 17.12.103. DateYear

---

### Category

**General-Purpose Functions**

### Description

Extract the year number from a date

### Syntax

DateYear(date)

### Parameters

date - an integer date value e.g. 20230816, S.Date, T.DateIn, etc.

## 17.12.104. Day

---

## Category

### Bar Data Values

## Description

Day of month of this bar

## Notes

Returns the day of month of the current bar date as a number.

Negative offsets, e.g. *Day[-5]*, can be legitimately used to obtain the day of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best results when future dates are required, a **HolidayList** should also be provided.

## 17.12.105. DayOfWeek

---

## Category

### Bar Data Values

## Description

Numeric code for day of week of this bar

## Notes

1=Monday

2=Tuesday

3=Wednesday

4=Thursday

5=Friday

6=Saturday (e.g. for Crypto)

7=Sunday (ditto)

Negative offsets, e.g. *DayOfWeek[-1]*, can be legitimately used to obtain the weekday of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best results when future dates are required, a **HolidayList** should also be provided.

## 17.12.106. DayOfYear

---

## Category

### Bar Data Values

## Description

Day of year of this bar

## Notes

Returns the calendar day of year, not the number of bars.

For example, a February 1 bar will always return 32. A December 31 bar will return 365 on normal years or 366 on leap years.

Negative offsets, e.g. *DayOfYear[-10]*, can be legitimately used to obtain the day of year of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best

results when future dates are required, a **HolidayList** should also be provided.

## 17.12.107. Days

---

### Category

#### General-Purpose Functions

### Description

Counts the number of calendar or market days from one date to another

### Syntax

Days(date1, date2, market {false})

### Parameters

date1 - formula returning a date

date2 - formula returning a date

market - optionally counts market days rather than the default calendar days

### Notes

The parameters can be date constants in YYYYMMDD format, or any function that returns a date, e.g. **BarDate** or **ToDate** or **InfoExpiry**.

One use of this function is to convert a bar count to a date count, e.g. *Days(BarDate[20], BarDate)*.

If you need to know the bar (market day) count between two dates, pass TRUE as the optional third argument.

## 17.12.108. DebugEntry

---

### Category

#### Strategy Elements

### Description

Log output from a running test at position entry (or skip) time

### Input

Any formula returning a **string** (blank string "" means don't log anything) or numeric value (0 means don't log anything)

### Notes

If this formula is present in a strategy, it will be evaluated for every setup just prior to position entry processing

If the *DebugEntry* formula returns a non-empty string, then whatever it returns is added to the **log window** text which is displayed at the end of the test run.

If the formula returns a numeric value, that number will be logged whenever it is non-zero.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (from *mr\_sample\_debug.rts*):

```
DebugEntry: if(debug_sym and long_setup, Format("limit={#2}, nextlow={#2}, miss={#2}", LongLimit, next_low, next_low - LongLimit), "")
```

## 17.12.109. DebugExit

---

## Category

### Strategy Elements

## Description

Log output from a running test at ExitRule evaluation time

## Input

Any formula returning a **string** (blank string "" means don't log anything)

## Notes

If this formula is present in a strategy, it will be evaluated for every stock every day of the test, just prior to the **ExitRule** formula evaluation.

If the *DebugExit* formula returns a non-empty string, then whatever it returns is added to the **log window** text which is displayed at the end of the test run.

If the formula returns a numeric value, that number will be logged whenever it is non-zero.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (from *mr\_sample\_debug.rts*):

```
DebugExit: if(debug_sym, Format("c={#2}, c[1]={#2}, held={#}", c, c[1], BarsHeld), "")
```

---

## 17.12.110. DebugTargetStop

## Category

### Strategy Elements

## Description

Log output from a running test at ExitLimit / ExitStop evaluation time

## Input

Any formula returning a **string** (blank string "" means don't log anything)

## Notes

If this formula is present in a strategy, it will be evaluated for every stock every day of the test, just prior to the **ExitLimit** and **ExitStop** formula evaluations.

If the *DebugTargetStop* formula returns a non-empty string, then whatever it returns is added to the **log window** text which is displayed at the end of the test run.

If the formula returns a numeric value, that number will be logged whenever it is non-zero.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (from *mr\_sample\_debug.rts*):

```
DebugTargetStop: if(debug_sym, Format("target={#2}, nexthigh={#2}, miss={#2}", long_target, next_high, long_target - next_high), "")
```

---

## 17.12.111. Dividend

## Category

### Bar Data Values

## Description

Dividend amount (\$/share) earned on an ex-dividend date

## Notes

Dividend will be 0 for every bar except for those of ex-dividend dates.

In other words, for any bar, Dividend is the \$/share you will receive if you held the stock that morning before the open.

Dividend values will only be present in the data if the data source used for import provided them.

**Norgate** and **Yahoo** both provide dividend amounts.

If **Adjustment** is set to *TotalReturn* when Norgate data is imported, then dividends are converted to splits (price adjustments) rather than reported as dividend payments.

When data includes dividend payments and a position is held across an ex-dividend date in a backtest, RealTest assumes the dividend was received on that date, adds its amount to total equity, and reports it as a separate item in the trade list.

The main reason you would reference the dividend amount directly would be to filter for stocks with high or low dividends, or use time until next ex-dividend date as a factor in a strategy.

## 17.12.112. DllDataCalc

---

### Category

#### General-Purpose Functions

### Description

Call a custom external DLL function to calculate a Data item

### Syntax

DllDataCalc(...)

### Parameters

none required, any number optional

### Notes

A native Windows DLL must be specifically built for this purpose.

See the RTDLL folder within your RealTest installation folder for details and an example.

## 17.12.113. EMA or XAvg

---

### Category

#### Multi-Bar Functions

### Description

Exponential Moving Average

### Syntax

EMA(expr, count) or XAvg(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *EMA* or *XAvg* can be used as the name of this function.

*Count* is usually thought of as an integer representing a number of bars, but can actually be any decimal value. An EMA is constructed by multiplying each difference between the result so far and the next value by a factor equal to  $2 / (\text{count} + 1)$ .

To calculate an EMA of *count* length with full precision, at least  $5 * count$  bars are required. RealTest calculates all exponential functions using however many bars are available, but if you require full precision, you will need to start your backtests that many bars after the start of your data file.

Note that, like most other backtesting software, RealTest begins each EMA calculation with  $MA(expr, count)$ , then begins to apply the EMA weighting for subsequent bars once *count* has been reached.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.114. EndDate (Import)

---

### Category

**Import Specification**

### Description

The last date to include in imported data

### Choices

**Date Constant** - a literal date

*Latest* - always use the most recent available date

## 17.12.115. EndDate (Setting)

---

### Category

**Settings**

### Description

The last date to include in a scan

### Choices

**Date Constant** - a literal date

*Latest* - always use the most recent available date

### Notes

If a date range is not specified in a script then the dates from the **Settings Panel** will be used.

To get the most recent N bars, use *EndDate: Latest* with **NumBars** but no **StartDate**.

## 17.12.116. EndOfMonth

---

### Category

**Bar Data Values**

### Description

Indicates whether this bar is the last bar of its month

### Notes

Returns 1 for the last bar of a month, otherwise 0.

Note that this function operates on the unified date list for all symbols in the current data file by default, or the date list of the **CalendarSym** if specified.

To check whether the current bar is the last bar of the month for a specific symbol, use *Month[-1]* <> *Month*.

To work correctly when generating tomorrow's orders or using negative (future) offsets, a **HolidayList** should also be provided.

## 17.12.117. EndOfWeek

---

### Category

**Bar Data Values**

### Description

Indicates whether this bar is the last bar of the market week

### Notes

Returns 1 for the last bar of a week (typically Friday), otherwise 0.

Note that this function operates on the unified date list for all symbols in the current data file by default, or the date list of the **CalendarSym** if specified.

To check whether the current bar is the last bar of the week for a specific symbol, use *Week[-1]* <> *Week*.

To work correctly when generating tomorrow's orders or using negative (future) offsets, a **HolidayList** should also be provided.

## 17.12.118. EntryDate

---

### Category

**Current Position Information**

### Description

The date on which this position was entered

### Notes

The return value is a number in YYYYMMDD format and is the date on which the entry occurred (not the *EntrySetup* signal day).

For a Weekly or Monthly strategy, this will be the actual entry date (e.g. a Monday if Weekly), not the date of the weekly entry bar, which is always its end date (typically a Friday).

If used as *Combined(EntryDate)* when multiple positions are open in the same symbol (whether due to pyramiding or multiple strategies), the most recent entry date for that symbol will be returned.

See also **BarsHeld**.

## 17.12.119. EntryLimit

---

### Category

**Strategy Elements**

### Description

Price to use when entering a position with a limit order

## Input

Any formula specifying a price per share

## Notes

The price returned by the *EntryLimit* formula is used to place a one-day limit order. For a long buy, the Low must be less than or equal to the limit price to potentially generate an entry. For a short sale, the High must be greater than or equal to the limit price.

If the opening price for a stock is at or beyond the limit price, then the the order is assumed to have been filled at the opening price, otherwise it fills at the limit price. Limit orders that fill at the open are assumed to have filled first when some entries have to be skipped due to position count or investment level caps.

If both *EntryLimit* and **EntryStop** are used, then the entry order becomes a stop+limit order. For this order to fill, both of these prices must be touched. The most common reason to use a stop+limit order is to avoid entries in a stop-based entry strategy where the opening gap is excessively large.

When a strategy includes an *EntryLimit* and/or an **EntryStop** formula, all entry-related formulas are evaluated using the *prior day* as the current bar, so that there can be no possibility of a look-ahead error. **EntrySetup** is evaluated first, and no other entry formulas are evaluated if it returns 0. (The only exception is when **EntryTime** is *ThisClose*, which implies that a live data feed would be used to enter just before the close.)

If the *EntryLimit* formula returns 0, this means "always enter", i.e., it becomes a market order (or a simple stop order if *EntryStop* was also specified).

## Examples

Enter long when price drops 4% below yesterday's close: *EntryLimit*:  $C * 0.96$

Enter short when price touches an upper Bollinger Band: *EntryLimit*:  $BBTop(20, 2)$

Enter long when RSI(2) reaches 5: *EntryLimit*:  $RRSI(2,5)$

## 17.12.120. EntryRank

---

### Category

**Current Position Information**

### Description

Returns the rank number for this position when **EntryScore** was evaluated at entry time.

### Notes

*EntryRank* can be referred to in any strategy formula except **EntrySetup**.

Entry ranks can also be observed by running a test with *TestOutput: Log* enabled.

## 17.12.121. EntryScore

---

### Category

**Strategy Elements**

### Description

Ranks potential entries when a strategy has more setups than can be entered

### Input

Any formula specifying a numeric value

## Notes

*EntryScore* is a **Legacy Mode** formula. When using the default top-down mode, always use **SetupScore** to rank your setups.

There is no longer a use case for *EntryScore* with top-down mode.

---

*The following notes apply only to legacy mode.*

Setups with higher scores are entered first.

If *EntryScore* is not specified, setups will be entered in alphabetical order by symbol.

The number of positions that a strategy can enter per day is determined by evaluating the **MaxSetups**, **MaxEntries**, **MaxExposure**, **MaxInvested**, and, **MaxPositions** formulas at entry time.

For strategies that include an **EntryLimit** or **EntryStop**, beware that use of an *EntryScore* formula constitutes a look-ahead error. In fact, the only reason this strategy element exists is to permit either random or specific entry order assumptions to be experimented with to get a feel for the range of possible results given that actual entry order cannot be known in advance.

For strategies that use limit or stop orders and want to cap the number of orders placed to the number you would actually want to be filled (without look-ahead bias), it is recommended to use top-down mode.

For more information on how the backtest engine works, see **Backtest Engine Details**.

## 17.12.122. EntrySetup

---

### Category

#### Strategy Elements

### Description

Determines whether a stock is eligible for entry

### Input

Any formula specifying a true/false condition (non-zero means true)

### Notes

*EntrySetup* is the first formula evaluated for each stock on each date of a backtest. If the return value is non-zero (true) then the stock is set up for entry (is an entry candidate). If a strategy has no *EntrySetup* formula, then it will produce no trades (unless it uses an **Imported Trade List**).

If a strategy also includes an **EntryLimit** and/or **EntryStop** formula, or if it specifies **EntryTime** as *NextOpen* (or leaves it unspecified), then all entry-related formulas including *EntrySetup* are evaluated using **the day prior to entry day** as the most recent bar in the formula. The only case where the entry-day bar can be accessed in entry-related formulas is when none of those things are true, i.e., when the strategy models entry with a market order just before the close.

For more information on how the backtest engine works, see **Backtest Engine Details**.

*EntrySetup* formulas can, at times, be rather complex. For this reason, it is highly recommended to make use of the **Data Section** for most of your entry condition logic. I will often have a series of data items that feed into a final one called something like "IsSetup", then just use *EntrySetup: IsSetup*.

The `mr_sample.rts` **sample script** includes an example of this tactic:

```

▼ Data:
  ATRx: ATR(5)
  EMaX: EMA(C,5)
  Liquid: C >= 20 and Avg(V, 20) >= 200000
  Volatile: ATRx > 0.03 * C
  Uptrend: C > Avg(C, Min(BarNum,150))
  Biotech: Top(Info(5),4) = 5620 // assumes Norgate data import
  Constituent: Index(1) // Norgate constituency flag
  Universe: Constituent and Liquid and Volatile and Uptrend and not(Biotech)

▼ Parameters:
  NumPos: from 5 to 20 step 5 def 10
  PctExt: from 1 to 5 step 0.5 def 2.5

▼ Strategy: mr_long // mean-reversion long strategy
  Using: base
  Side: Long
  EntrySetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMaX) // oversold
  EntryLimit: C - ATRx // drops another ATR
  ExitRule: C > C[1] or BarsHeld > 5 // sell on first up day or after 5 days

```

## 17.12.123. EntrySkip

---

### Category

#### Strategy Elements

### Description

Enables skipping an entry if a condition applies

### Input

Any formula specifying a true/false condition (non-zero means true)

### Notes

After all other conditions for entry have been met, the *EntrySkip* formula is evaluated (if provided). If the result is non-zero (true) or if the formula can't be evaluated (nan) then the entry is not taken.

All skipped entries (potential entries that passed the **EntrySetup** condition but did not become positions for any reason) are optionally included in the **Trade List** when a test is run. Skipped entries shown in the trade list include a column where the reason the entry was skipped is shown. Trades skipped because the *EntrySkip* condition was true show "skip formula" as their skip reason.

A good use of *EntrySkip* is, for example, *EntrySkip: random() < 0.05*. This would randomly skip about 5% of entries, e.g. to simulate not being able to borrow shares to short. By running this same test a number of times we can see the probable range of how this would impact the stats of the strategy.

## 17.12.124. EntryStop

---

### Category

#### Strategy Elements

### Description

Price to use when entering a position with a stop order

### Input

Any formula specifying a price per share

### Notes

The price returned by the *EntryStop* formula is used to place a one-day stop order. For a long buy,

the High must be greater than or equal to the stop price to potentially generate an entry. For a short sale, the Low must be less than or equal to the stop price.

If the opening price for a stock is at or beyond the stop price, then the the order is assumed to have been filled at the opening price, otherwise it fills at the stop price. Stop orders that fill at the open are assumed to have filled first when some entries have to be skipped due to position count or investment level caps.

If both **EntryLimit** and *EntryStop* are used, then the entry order becomes a stop+limit order. For this order to fill, both of these prices must be touched. The most common reason to use a stop+limit order is to avoid entries in a stop-based entry strategy where the opening gap is excessively large.

When a strategy includes an **EntryLimit** and/or an *EntryStop* formula, all entry-related formulas are evaluated using **the prior day** as the current bar, so that there can be no possibility of a look-ahead error. **EntrySetup** is evaluated first, and no other entry formulas are evaluated if it returns 0. (The only exception is when **EntryTime** is *ThisClose*, which implies that a live data feed would be used to enter just before the close.)

If the *EntryStop* formula returns 0, this means "always enter", i.e., it becomes a market order (or a simple limit order if *EntryLimit* was also specified).

## Examples

Enter long when price moves above yesterday's high: *EntryStop: H + TickSize*

Enter short when price moves below a moving average: *EntryStop: MA(C,20) - TickSize*

Enter long if price moves above the 10-day highest close: *EntryStop: Highest(C,10) + TickSize*

Add a limit order to the prior example to avoid entry at more than 2% above the stop price:  
*EntryLimit: 1.02 \* Highest(C,10)*

## 17.12.125. EntryTime

---

### Category

#### Strategy Elements

### Description

Specifies when a strategy enters new positions

### Choices

*ThisClose* - entries occur at the close of the current day

*Intraday* - entries occur when a trigger price is first touched tomorrow (default for limit and/or stop orders)

*NextOpen* - entries occur at tomorrow's open (default for market orders)

*NextClose* - entries occur at tomorrow's close

### Notes

Using *ThisClose* implies an ability to generate realtime trading signals using live data. RealTest will not be able to generate **Tomorrow's Orders** for *ThisClose* entries.

For strategies with no **EntryLimit** or **EntryStop**, *EntryTime* controls the time at which a market order is generated. *NextOpen* implies that a standard **MKT** order is placed before the open, to be filled at the open. (*Intraday* is interpreted as *NextOpen* for market orders.) *NextClose* implies that a **MOC** order is placed before the open, to be filled at the close.

*EntryTime* also applies to strategies that use an *EntryLimit* and/or *EntryStop* price.

Using *ThisClose* entry with *EntryLimit* specified is equivalent to adding "and C < n" to **EntrySetup** (for a long-side strategy) where "n" is the *EntryLimit* price. Similarly, a long *EntryStop* would be equivalent to "and C > n". Reverse these comparison operator directions for short-side strategies. As previously stated, orders cannot be generated in advance for this mode.

With *Intraday*, *NextOpen*, or *NextClose* limit or stop entries, the trigger price is calculated using the prior bar, and the order can be placed in advance.

A *Intraday EntryLimit* or *EntryStop* is a standard **LMT DAY** or **STP DAY** order.

A *NextOpen EntryLimit* is a **LMT OPG** order, and a *NextOpen EntryStop* is **STP GTD 9:31 EST**.

A *NextClose EntryLimit* is a **LOC** order, and a *NextClose EntryStop* is **STP GAT 15:58 EST**.

In all of the above, if both *EntryLimit* and *EntryStop* are specified, the order becomes **STPLMT** with the same qualifiers.

(All of the above order types are stated using Interactive Brokers order nomenclature, and can be automatically generated by adding *TestOutput: Orders* and *OrdersTemplate:*

*Examples\ib\_basket\_trader.csv* to your **Settings**.)

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

## 17.12.126. EntryTradeValue

---

### Category

#### Strategy Elements

### Description

Calculates a value to store in **T.ValueIn** item in the trade list record for this entry

### Input

Any formula specifying a numeric value

### Notes

This item can be useful in **Trade Statistics Functions**, especially when applied to **Results** formulas which do not support access to bar data values or indicator functions.

## 17.12.127. ESD

---

### Category

#### Multi-Bar Functions

### Description

Exponential Standard Deviation

### Syntax

ESD(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

This function calculates a standard deviation of values as an exponential series using a constant weighting factor represented as a bar count.

As in the **EMA** function, count is converted to a weight factor using the formula  $factor = 2 / (count + 1)$ .

This factor is then used to calculate two intermediate series for an array of values.

An EMA is calculated in place of the *mean* in a typical **StdDev** calculation.

The average of the squared differences between each value and the mean (i.e. the standard deviation) is also calculated as an EMA using the same weight factor.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.128. Event

---

### Category

**Bar Data Values**

### Description

Get the event code for this bar or the most recent event code

### Syntax

Event(type, latest {optional})

### Parameters

type - event type number

latest - most-recent (1) vs. this-bar-only (0) flag -- 0 is the default

### Notes

If the current data file includes a user-defined **Event List**, this function looks up an event code for the current symbol and bar.

If an event is found that has *type* as its type number, then its value is returned.

If *latest* is true (non-zero), then the most recent event of this type for this symbol prior to (or on) the current bar is returned.

If no matching event is found then 0 is returned.

An event is considered to have occurred "on a bar" if the event's time value (from the imported event list)  $\leq$  16:00 (160000).

## 17.12.129. EventListFile

---

### Category

**Import Specification**

### Description

Path to a CSV file containing a list of specific event data to include in imported bar data

### Input

File path

Notes

See **Event** and **The Event List File** for details on how this mechanism works and can be used.

## 17.12.130. ExchangeMap

---

### Category

**Settings**

## Description

Specifies the path to a CSV file which helps RealTest generate orders with correct symbols, exchange names, and GAT/GTD times

## Notes

An exchange map is required in order for RealTest to generate orders in any **OrdersMode** other than *Text*.

If an *ExchangeMap* **Setting** is not specified, RealTest looks for a file called *ExchangeMap.csv* in the following locations:

1. the current **OrderClerkFolder** when *OrdersMode* is *OrderClerk*
2. the RealTest installation folder

The default *ExchangeMap.csv* file installed with RealTest looks like this:

Exchange	Primary	Match1	Match2	HasMoc	TimeZone	LastMocTime	NearCloseTime	EarlyLastMocTime	EarlyNearCloseTime	LateLocTime	WeeklyLastMocTime	LastOpenTime
BVME.ETF	BVME.ETF	*.MI		0	MET	15:58:00	15:58:00				15:58:00	9:01:00
EBS	EBS	*.MI		0	MET	15:58:00	15:58:00				15:58:00	10:05:00
IBIS2	IBIS2	*.MI		0	MET	15:58:00	15:58:00				15:58:00	9:01:00
AMS	AEB	*.AS		0	MET	17:38:00	17:38:00				17:38:00	9:01:00
ASX	ASX	*.AX	*.AU	0	Australia/NSW	15:58:00	15:58:00			16:02:00	11:00:00	10:10:00
GER	IBIS	*.DE	*.BN	0	MET	16:58:00	16:58:00				16:58:00	9:01:00
PAR	SBF	*.PA		0	MET	17:28:00	17:28:00				17:28:00	9:01:00
LSE*	LSE	*.L		0	MET	16:28:00	16:28:00				16:28:00	9:01:00
TS*	TSE	*.CA	*.TO	1	US/Eastern	15:40:00	15:58:00				11:00:00	9:31:00
CSE	CSE			0	US/Eastern	15:58:00	15:58:00				11:00:00	9:31:00
NEO	NEO			0	US/Eastern	15:58:00	15:58:00				11:00:00	9:31:00
CBOE	CBOE			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
CBOT	CBOT			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
CME	CME			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
COMEX	COMEX			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
NYBOT	NYBOT			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
NYMEX	NYMEX			0	US/Eastern	15:58:00	15:58:00				15:58:00	9:31:00
*	AMEX			1	US/Eastern	15:40:00	15:58:00	12:40:00	12:58:00		15:40:00	9:31:00

The column names and sequence must not be changed.

The content and purpose of each column is described below:

Column	Content	Purpose
Exchange	exchange name with optional * or ? <b>Match</b> wildcard characters	matches the <b>?Exchange</b> of the stock for which an order is being generated
Primary	brokerage exchange	specifies the primary exchange in SMART/primary
Match1	symbol matching pattern	if the stock has no exchange specified, matches the symbol pattern to determine the exchange to use
Match2	same as above	alternative symbol pattern (multiple data sources)
HasMoc	0 or 1	1 if this exchange supports true MOC orders else 0
TimeZone	Exchange timezone name in IANA <b>tz database</b> format	Appended to all date+time order fields for this exchange (required by IB)
LastMocTime	24-hour hh:mm:ss	"Good Until" time for entry orders for same-day MOC exit
NearCloseTime	same as above	"Good After" time for "ThisClose" or "NextClose" entry or exit MKT orders including MOC if not supported by exchange (date determined automatically)

EarlyLastMocTime	same as above	LastMocTime for early-close days
EarlyNearCloseTime	same as above	NearCloseTime for early-close days
LateLocTime	same as above	closing auction LOC placement time
LastOpenTime	same as above	GTD time for "at-open" STP orders

## 17.12.131. ExcludeIf

---

### Category

#### Import Specification

### Description

Import filter formula (exclude symbol if true)

### Input

number

## 17.12.132. ExcludeList

---

### Category

#### Import Specification

### Description

List of one or more symbols to exclude when **importing data**

### Notes

An **Import Section** can have any number of *ExcludeList* statements.

Each *ExcludeList* statement can take one of the following forms:

- One or more symbols, separated by commas (up to a maximum of 260 characters in total)
- A path to a local file containing a list of symbols

If a symbol from any **IncludeList** also appears on any *ExcludeList*, then that symbol will not be included in the imported data.

Use of an *ExcludeList* is a convenient way to omit a few specific symbols from a data file without having to modify an existing watchlist.

## 17.12.133. ExitLimit

---

### Category

#### Strategy Elements

### Description

A price at which to exit a position using a limit order (profit target)

### Input

Any formula specifying a price per share

### Notes

*ExitLimit* specifies a "target" price, implemented by default as a **LMT DAY** order. The formula is re-evaluated each day, so it can function as either a static or moving target, depending on how it is specified.

*ExitLimit* is always evaluated using **the day prior to exit day** as the most recent bar in the formula.

A simple fixed 5% profit target could be specified as  $FillPrice * 1.05$  for a long position or  $FillPrice * 0.95$  for a short position.

An ATR-based target might be expressed as  $FillPrice + 2 * ATR(14)$  for a long position or  $FillPrice - 2 * ATR(14)$  for a short position.

A strategy can include any combination of an **ExitRule**, *ExitLimit* and/or **ExitStop** (stop-loss). Whichever one is determined to have occurred first will be the one used in the test. The type of exit that occurred for each trade is displayed in the *Reason* column of the **Trade List**.

See also **ExitLimitTime** for other target-order alternatives.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

## 17.12.134. ExitLimitQty

---

### Category

#### Strategy Elements

### Description

Specifies the number of shares or contracts to buy or sell when a position exits on an **ExitLimit**

### Input

Any formula specifying a number of shares or contracts

### Notes

*ExitLimitQty* can be used to model partial limit (target) exits (scaling out of positions or reducing their size).

If *ExitLimitQty* is not specified then the entire position is always exited when an *ExitLimit* triggers.

The *ExitLimitQty* formula must calculate the number of shares or contracts to exit regardless of the strategy's **QtyType** setting.

To model partial exits for the other exit types, **ExitQty** and/or **ExitStopQty** can also be added to a strategy.

See **ExitQty** for additional information and an example.

## 17.12.135. ExitLimitTime

---

### Category

#### Strategy Elements

### Description

Specifies how a strategy exits positions when the **ExitLimit** formula is used

### Choices

*ThisClose* - exit at today's close if today's closing price touches or exceeds the limit price (impractical)

*Intraday* - *ExitLimit* specifies the price of a live limit order to be placed tomorrow (default)

*NextOpen* - exit at tomorrow's open if today's close touched or exceeded the limit price

*NextClose* - exit at tomorrow's close if tomorrow's close touches or exceeds the limit price (calculated from today's bar)

### Notes

Using *ThisClose* with *ExitLimit* equivalent to adding "and C > n", where "n" is the *ExitLimit* price (for

a long-side strategy), to the **ExitRule** of a strategy with **ExitTime** *ThisClose*. In either case, orders could not be generated in advance for this mode.

With *Intraday*, *NextOpen*, or *NextClose* *ExitLimit* orders, the trigger price is calculated using the prior bar, and the order can be placed in advance.

A *Intraday* *ExitLimit* is a standard **LMT DAY** order.

A *NextOpen* *ExitLimit* is a **MKT** order that is placed if the prior close touched the limit price.

A *NextClose* *ExitLimit* is a **LOC** (limit on close) order at the specified limit price.

Note that the *Exit Logic* of a strategy works differently from its *Entry Logic*. Specifically, there is always only one *Entry* order, while there can be up to three *Exit* orders. The three exit order types each have their own time specifier and function as a "one-cancels-all" order bracket, where the order to logically trigger first becomes the position exit and the others do not.

To implement a trailing limit price, use **PrevExitLimit** in the *ExitLimit* price calculation.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

## 17.12.136. ExitNum

---

### Category

**Current Position Information**

### Description

Returns the number of this exit for this position

### Notes

*ExitNum* will normally be 1 unless **ExitQty**, **ExitLimitQty** or **ExitStopQty** was previously used to exit part of the position.

*ExitNum* can be used in any exit-related strategy formula as part of its exit logic.

## 17.12.137. ExitQty

---

### Category

**Strategy Elements**

### Description

Specifies the number of shares or contracts to buy or sell when **ExitRule** is true

### Input

Any formula specifying a number of shares or contracts

### Notes

*ExitQty* can be used to model partial exits (scaling out of positions or reducing their size).

If *ExitQty* is not specified then the entire position is always exited when *ExitRule* is true.

The *ExitQty* formula must calculate the number of shares or contracts to exit regardless of the strategy's **QtyType** setting.

To model partial exits for the other exit types, **ExitLimitQty** and/or **ExitStopQty** can also be added to a strategy.

If the specified quantity is less than or equal to 0 then the exit is cancelled (the position is held at its current size).

If the quantity is greater than or equal to the current position size then the entire position is exited.

Use **FillQty** in these formulas to express the partial exit in terms of the original position size.

For example to exit one fourth of the original position on each of the first four days after entry:

```
ExitRule:      1
ExitQty:      FillQty / 4
```

This use of *FillQty* rather than **Shares** is necessary because *Shares* will be reduced after each partial exit.

Partial exits are implemented internally by temporarily cloning the position, changing the clone's quantity to the desired exit quantity, and then processing the exit of that new position. The quantity of the original position (*Shares*) is then changed to the prior quantity minus the exit quantity.

When looking at the **trade list** of a strategy that uses partial exits, none of the trades will show the original entry quantity as their *QtyIn* values. Instead *QtyIn* is always equal to *QtyOut* (the partial exit quantity) for each individual trade (unless it was held across a split).

To see the original entry quantities in the trade list, add this to the strategy:

```
EntryTradeValue:  FillQty
```

and add this to **Trades.rts**:

```
ValueIn:          T.ValueIn
```

## 17.12.138. ExitRank

---

### Category

**Current Position Information**

### Description

Returns the rank number for this position based on **ExitScore**

### Notes

*ExitScore* is recalculated each bar for each open position, before any of the exit formulas are evaluated.

Positions are then sorted by this score and their resulting rank numbers are stored.

Use *ExitRank* to access the rank number of the current position based on that evaluation.

## 17.12.139. ExitRule

---

### Category

**Strategy Elements**

### Description

Specifies one or more conditions that would trigger a position exit

### Input

Any formula specifying a true/false condition (non-zero means true)

### Notes

The *ExitRule* for a strategy is evaluated for every position every day. If it ever returns true (non-zero), the position is exited either that day at the close or the next day at the open, depending on the **ExitTime** setting.

When exit time is *NextOpen* (the default) or *NextClose*, *ExitRule* is evaluated using **the day prior to exit day** as the most recent bar in the formula. For *ThisClose* exits, the exit day is the most recent bar.

This formula is typically used to specify a conditional exits such as  $C > C[1]$ , or time-based exits such as *BarsHeld=10*.

A strategy can include any combination of an *ExitRule*, **ExitLimit** (target) and/or **ExitStop** (stop-loss). Whichever one is determined to have occurred first will be the one used in the test. The type of exit that occurred for each trade is displayed in the *Reason* column of the **Trade List**.

Optionally, for *ExitRule* formulas with multiple "or" conditions, the formula can return a **string** which both serves as the "true" signal of the condition and provides the *Reason* name for the exit in the trade list.

This example shows how to structure a multi-reason *ExitRule* formula using the **Select** function, and how each reason string is shown in the Trades Window:

Active Script - C:\RealTest\Examples\mr\_sample.rts

```

Strategy: mr_long // mean-reversion long strategy
Using: base
Side: Long
EntrySetup: Universe and C < (1 - PctExt / 100) * Min(O, C[1], EMA5) // oversold
EntryLimit: LongLimit
ExitLimit: FillPrice * (1 + Target/100) // intraday profit target
ExitRule: select(C > C[1], "up day", BarsHeld == 5, "time stop") // sell on first up day or after 5 days

```

Trade List - Test 1 - 7,765 Trades

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn	DateOut	TimeOut	QtyOut	PriceOut	Reason	Bars
00209	mr_long	ANN-201508	Long	5/17/10	intraday	535	21.38	5/24/10	open	535	21.87	up day	5
00210	mr_long	TEN	Long	5/17/10	intraday	526	21.73	5/24/10	open	526	20.59	up day	5
00211	mr_long	TEX	Long	5/17/10	intraday	486	23.54	5/24/10	open	486	21.09	up day	5
00212	mr_long	USG-201904	Long	5/17/10	intraday	569	20.09	5/24/10	open	569.00	17.19	time stop	5
00213	mr_long	UFS	Long	5/18/10	intraday	190	61.64	5/24/10	open	190	59.53	up day	4

Note that when used this way, the *Select* statement must be the entire *ExitRule* formula.

For example, to only check a pair of exit rules only at the end of each month, you must use this:

*ExitRule: Select(EndOfMonth and condition1, "reason1", EndOfMonth and condition2, "reason2")*

rather than this:

*ExitRule: EndOfMonth and Select(condition1, "reason1", condition2, "reason2")*

It does **not** work to move this special type of *Select* statement to a Library or Data item.

For more information on how the backtest engine works, see **Backtest Engine Details**.

## 17.12.140. ExitScore

### Category

#### Strategy Elements

### Description

Calculates a score value for use in ranking open positions prior to evaluating exit criteria each day in a test

### Input

Any formula specifying a numeric value

### Notes

*ExitScore* is recalculated each bar for each open position, before any of the exit formulas are evaluated.

Positions are then sorted by this score and their resulting rank numbers are stored.

Use **ExitRank** to access the rank number of the current position based on that evaluation.

## 17.12.141. ExitStop

---

### Category

#### Strategy Elements

### Description

A price at which to exit a position using a stop order (stop loss)

### Input

Any formula specifying a price per share

### Notes

*ExitStop* specifies a "stop" price, implemented as a **STP DAY** order. The formula is re-evaluated each day, so it can function as either a static or trailing stop, depending on how it is specified.

*ExitStop* is always evaluated using **the day prior to exit day** as the most recent bar in the formula.

A simple fixed 5% stop loss could be specified as  $FillPrice * 0.95$  for a long position or  $FillPrice * 1.05$  for a short position.

A trailing stop 5% below the highest high since entry could, for a long position, be expressed as  $0.95 * Highest(H, BarsHeld)$ .

A strategy can include any combination of an **ExitRule**, **ExitLimit** (target) and/or *ExitStop*. Whichever one is determined to have occurred first will be the one used in the test. The type of exit that occurred for each trade is displayed in the *Reason* column of the **Trade List**.

See also **ExitStopTime** for other stop-order alternatives.

For more information on how the backtest engine works, see **Backtest Engine Details**.

## 17.12.142. ExitStopQty

---

### Category

#### Strategy Elements

### Description

Specifies the number of shares or contracts to buy or sell when a position exits on an **ExitStop**

### Input

Any formula specifying a number of shares or contracts

### Notes

*ExitStopQty* can be used to model partial stop exits (scaling out of positions or reducing their size).

If *ExitStopQty* is not specified then the entire position is always exited when an *ExitStop* triggers.

The *ExitStopQty* formula must calculate the number of shares or contracts to exit regardless of the strategy's **QtyType** setting.

To model partial exits for the other exit types, **ExitQty** and/or **ExitLimitQty** can also be added to a strategy.

See **ExitQty** for additional information and an example.

## 17.12.143. ExitStopTime

---

### Category

## Strategy Elements

### Description

Specifies how a strategy exits positions when the **ExitStop** formula is used

### Choices

*ThisClose* - exit at today's close if today's closing price touches or exceeds the stop price (impractical)

*Intraday* - *ExitStop* specifies the price of a live stop order to be placed tomorrow (default)

*NextOpen* - exit at tomorrow's open if today's close touched or exceeded the stop price

*NextClose* - exit at (or just before) tomorrow's close if tomorrow's close touches or exceeds the stop price (calculated from today's bar)

### Notes

Using *ThisClose* with *ExitStop* equivalent to adding "and  $C < n$ ", where "n" is the *ExitStop* price (for a long-side strategy), to the **ExitRule** of a strategy with **ExitTime** *ThisClose*. In either case, orders could not be generated in advance for this mode.

With *Intraday*, *NextOpen*, or *NextClose* *ExitStop* orders, the trigger price is calculated using the prior bar, and the order can be placed in advance.

A *Intraday* *ExitStop* is a standard **STP DAY** order.

A *NextOpen* *ExitStop* is a **MKT** order that is placed if the prior close touched the stop price.

A *NextClose* *ExitStop* is a **STP** order with a "good after time" clause with time a minute or two before the market closes.

Note that the *Exit Logic* of a strategy works differently from its *Entry Logic*. Specifically, there is always only one *Entry* order, while there can be up to three *Exit* orders. The three exit order types each have their own time specifier and function as a "one-cancels-all" order bracket, where the order to logically trigger first becomes the position exit and the others do not.

To implement a trailing stop price, use **PrevExitStop** in the *ExitStop* price calculation.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

## 17.12.144. ExitTime

---

### Category

#### Strategy Elements

### Description

Specifies when a strategy exits positions at market when the **ExitRule** condition is true (non-zero)

### Choices

*ThisClose* - exits occur at the close of the current day

*NextOpen* - exits occur at tomorrow's open (default)

*NextClose* - exits occur at tomorrow's close

### Notes

Using *ThisClose* implies an ability to generate realtime trading signals using live data. RealTest will not be able to generate **Tomorrow's Orders** for *ThisClose* exits.

*ExitTime* controls the time at which a market order is generated. *NextOpen* implies that a standard **MKT** order is placed before the open, to be filled at the open. *NextClose* implies that a **MOC** order is placed before the open, to be filled at the close.

For more information on how the backtest engine works in general, see [Backtest Engine Details](#).

## 17.12.145. ExitTradeValue

---

### Category

**Strategy Elements**

### Description

Calculates a value to store in **T.ValueOut** item in the trade list record for this exit

### Input

Any formula specifying a numeric value

### Notes

This item can be useful in **Trade Statistics Functions**, especially when applied to **Results** formulas which do not support access to bar data values or indicator functions.

## 17.12.146. Exp

---

### Category

**General-Purpose Functions**

### Description

Exponential function ( $e^x$ )

### Syntax

Exp(value)

### Parameters

value - formula

## 17.12.147. Extern

---

### Category

**General-Purpose Functions**

### Description

Evaluate for a different stock/contract or strategy or bar size

### Syntax

Extern(item\_reference, expression)

### Parameters

item\_reference - a stock symbol, a strategy name, or a bar size

expression - formula

### Notes

The default context in all formulas is the current stock for bar data items and the current strategy for daily stats items.

*Extern* allows you temporarily change context to a different stock or strategy or bar size.

To reference a stock, use a \$ to prefix the symbol of the desired stock, e.g. *Extern(\$MSFT, C)*

To reference a **Strategy**, **Benchmark** or **StatsGroup**, use a @ to prefix the name of the desired strategy, e.g. *Extern(@mr\_short, S.Equity)*

To calculate a formula using a specific bar size, use a ~ to prefix the name of the bar size, e.g. *Extern(~Weekly, MA(C, 20))*

Note that the expression referenced can be any formula, and it is most efficient to do as much as possible in a single *Extern* function. For example, *Extern(\$SPY, C > MA(C,20))* would be preferable to *Extern(\$SPY,C) > Extern(\$SPY, MA(C,20))* or, even worse, *Extern(\$SPY,C) > MA(Extern(\$SPY,C), 20)* -- though all would return the same result.

If a stock symbol already starts with a \$, e.g. \$SPX, you will have to add another \$, hence *Extern(\$\$SPX, C)*. See **Symbol Constants** for more information about these.

There are also special types of *Extern* references available for **individual futures contracts** and **corresponding industry index** symbols.

It is also possible to refer to a dynamic (non-constant) external symbol, either by string or number, using **SymNum**.

## 17.12.148. Extra

---

### Category

#### Bar Data Values

### Description

Turnover value from Norgate import, or user-defined Extra bar value from CSV import

### Notes

Norgate's *Turnover* value is not simply *Close \* Volume*. Rather, it is the sum of *Price \* Quantity* of all intraday trades (ticks) in a day. As such, true daily VWAP (volume-weighted average price) can be calculated as *Extra / Volume* when data is from Norgate.

With CSV import this field, which can also be referred to as *Extra*, will contain whatever values were included in the column defined as *Extra* in your CSV import.

## 17.12.149. F.xxx / F.xxx.Date

---

### Category

#### Stock/Contract Information

### Description

Value or report date of a **current** fundamental item that was imported from **Norgate**

### Notes

Norgate current fundamentals can optionally be added to imported data by adding **Fundamentals** to your **Import** definition and specifying one or more fundamental items to obtain.

Replace **xxx** with the name of an imported fundamental item, e.g. *F.epsactualq*, to access a value thus imported.

Use **F.xxx.Date**, e.g. *F.epsactualq.Date*, to obtain the date on which this item was reported,

according to Norgate.

The **Example Script** *fundamentals.rts* shows how this all works:

The screenshot shows the Active Script editor for *fundamentals.rts*. The script content is as follows:

```
// how to import current fundamental fields from Norgate and view them in a scan
// use auto-complete in the Fundamentals: statement to see all the available values

▽ Import:
  DataSource: Norgate
  IncludeList: .Dow Jones Industrial Average
  StartDate: 1/1/22
  EndDate: Latest
  SaveAs: temp.rtd
  Fundamentals: aepsnorm, arevps, abvps

▽ ScanSettings:
  EndDate: Latest
  NumBars: 1

▽ Scan:
  eps: F.aepsnorm
  revs: F.arevps
  book: F.abvps
  reported: F.aepsnorm.Date {/**}

// do not use these in backtesting -- they are current values only, not historical series
```

Two red callout boxes highlight specific parts of the script: "items to import" points to the `Fundamentals: aepsnorm, arevps, abvps` line, and "imported item values" points to the `eps: F.aepsnorm`, `revs: F.arevps`, and `book: F.abvps` lines.

Below the script is a table titled "Scan - 30 Items" showing the results of the scan:

Date	Symbol	eps	revs	book	reported
2/28/22	AAPL	5.61	21.69	3.84	9/25/21
2/28/22	AMGN	10.28	45.34	12.00	12/31/21
2/28/22	AXP	10.02	54.23	29.14	12/31/21
2/28/22	BA	-5.07	106.00	-25.47	12/31/21
2/28/22	CAT	11.89	92.93	30.76	12/31/21
2/28/22	CRM	4.49	22.85	45.15	1/31/21
2/28/22	CSCO	2.67	11.76	9.79	7/31/21
2/28/22	CVX	8.77	81.84	72.06	12/31/21

## 17.12.150. FillFraction

### Category

**Current Position Information**

### Description

Equity fraction of current position size at time of initial entry order (or actual entry)

### Notes

*FillFraction* returns the initially ordered position size as a fraction of `S.Alloc` at the time of the order.

This was the same fraction used to check the potential entry against the **MaxExposure** constraint.

*FillFraction* is the same as the actual position size fraction unless a gap causes a difference between order price and entry price.

If **QtyPrice** is *FillPrice* rather than the default *OrderPrice* or if the test was run in **LegacyMode** then *FillFraction* will always be the entry price.

The *FillFraction* of each trade becomes the **T.Fraction** value in the trade list.

The **S.Exposure** and **S.Usage** stats are the sum of *FillFraction* values of open positions each day, so they also will depend on how *FillFraction* is calculated.

## 17.12.151. FillPrice

---

### Category

#### Current Position Information

### Description

Entry fill price (or exit fill price when calculating commission or slippage for exit transactions)

### Notes

*FillPrice* is generally the price per share at which this position was entered. This will match the *PriceIn* value from the backtest **Trade List**.

In formulas evaluated at the same time as **EntrySetup** -- before tomorrow's entry price could have been known -- *FillPrice* returns **OrderPrice** by default.

This automatic substitution of *OrderPrice* for *FillPrice* typically applies to:

- **Quantity**
- **ExitLimit** and/or **ExitStop** for exit orders attached to an entry order

The purpose of this automatic substitution is to ensure that a future backtest will match the orders that would have been generated and placed before the market open.

On all subsequent days that a position is held, references to *FillPrice* return the actual entry price that was modeled.

This entry-day substitution allows *FillPrice* to be used in your *ExitLimit* and *ExitStop* formulas without needing extra logic to use *OrderPrice* on entry day and *FillPrice* thereafter.

If you would prefer that *FillPrice* always return the actual entry price (even for orders placed at the same time as the entry order when it could not have been known yet) add **QtyPrice: FillPrice** to the strategy definition.

The one exception to *FillPrice* being the *entry* price (or order price) is when it is used in the **Commission** or **Slippage** formulas.

Since those formulas are evaluated separately for the entry and exit sides of each round-trip trade, *FillPrice* automatically retrieves the *exit* price when these two formulas are evaluated at position exit time.

## 17.12.152. FillPriceAvg

---

### Category

#### Current Position Information

### Description

Average entry fill price of current position when pyramiding (**MaxSameSym** > 1)

### Notes

For pyramiding strategies, *FillPriceAvg* returns the volume-weighted average price of all entry transactions in the current position.

For non-pyramiding strategies, *FillPriceAvg* simply returns **FillPrice**.

## 17.12.153. FillPriceMax

---

## Category

### Current Position Information

## Description

Highest entry fill price of current position when pyramiding (**MaxSameSym** > 1)

## Notes

For pyramiding strategies, *FillPriceMax* returns the highest price of all entry transactions in the current position.

For non-pyramiding strategies, *FillPriceMax* simply returns **FillPrice**.

## 17.12.154. FillPriceMin

---

## Category

### Current Position Information

## Description

Lowest entry fill price of current position when pyramiding (**MaxSameSym** > 1)

## Notes

For pyramiding strategies, *FillPriceMin* returns the lowest price of all entry transactions in the current position.

For non-pyramiding strategies, *FillPriceMin* simply returns **FillPrice**.

## 17.12.155. FillQty

---

## Category

### Current Position Information

## Description

Shares or contracts in current position at time of entry

## Notes

For most positions *FillQty* will be the same as **Shares**.

The only time these quantities can differ is after a partial exit.

Partial exits can be specified using **ExitQty**, **ExitLimitQty**, and/or **ExitStopQty**.

Indeed the main purpose of *FillQty* is to facilitate expression of partial exit quantities in the above formulas.

For example to exit one fourth of the original position on each of the first four days after entry, simply write:

```
ExitRule:      1
ExitQty:      FillQty / 4
```

## 17.12.156. FillValue

---

## Category

### Current Position Information

## Description

Dollar value of current position at time of entry

## Notes

*FillValue* returns the initial position size, in notional dollars, of the current position.

This same value could be simply calculated as  $FillPrice * Shares * PointValue$ .

The one exception to *FillValue* being the initial position size is when it is used in the **Commission** or **Slippage** formulas. Since those formulas are evaluated separately for the entry and exit sides of each round-trip trade, *FillValue* automatically retrieves the *exit* position size when these two formulas are evaluated at position exit time.

If you need *FillValue* to be negative for short positions, multiply it by **Side**.

## 17.12.157. FilterNum

---

### Category

**Stock/Contract Information**

### Description

Filter formula number which the current stock passed on the current date during scan evaluation

### Notes

This syntax element is only applicable within a scan item definition. Scans with multiple filter formulas will loop through all the stocks on each date once for each provided filter, and include any symbol once for each filter it passes. *FilterNum* can then be used in the other scan output formulas to know which filter was just passed.

See **Multi-Row Scan** for details and an example.

## 17.12.158. Format

---

### Category

**String Functions**

### Description

Format a string with embedded items

### Syntax

`Format("string", ...)`

### Parameters

string - a **literal string** or **string function** result

... - any number of comma-separated formulas corresponding to format codes embedded in the format string

### Notes

The string parameter can contain one or more embedded **Format Specifiers**. Each one will correspond to a parameter passed to the function. The corresponding parameter (formula) is evaluated and its result is inserted in the string in place of the format specifier.

An example this would be `Format("5-day ROC {%2}", c / c[5] - 1)`.

Use `{?}` to insert a string within another string. The parameter that corresponds with `{?}` can be either a literal string or a function that returns a string. You can even write nested Format functions if you can think of a reason to do so.

## 17.12.159. Fundamentals

### Category

#### Import Specification

### Description

List of one or more **current** fundamental items to include when importing **Norgate** data

### Notes

Norgate provides more than 160 current fundamental data items for each stock.

To see the names and descriptions of these fields, add "Fundamentals: " to your Import section and then press F2 or start typing something. The auto-complete mechanism will guide you. Another way to quickly view all of the available fundamental items for a stock is to open a **Chart** and then select *Get Information* from the **Chart Menu**.

To access the items thus imported, use **F.xxx** (where xxx is the item name) to get the value or **F.xxx.Date** to get its release date.

The **Example Script** *fundamentals.rts* shows how this all works:

The screenshot shows the Active Script editor with the following script content:

```
// how to import current fundamental fields from Norgate and view them in a scan
// use auto-complete in the Fundamentals: statement to see all the available values

▽ Import:
  DataSource: Norgate
  IncludeList: .Dow Jones Industrial Average
  StartDate: 1/1/22
  EndDate: Latest
  SaveAs: temp.rtd
  Fundamentals: aepsnorm, arevps, abvps

▽ ScanSettings:
  EndDate: Latest
  NumBars: 1

▽ Scan:
  eps: F.aepsnorm
  revs: F.arevps
  book: F.abvps
  reported: F.aepsnorm.Date {/**}

// do not use these in backtesting -- they are current values only, not historical series
```

Two red callout boxes highlight specific parts of the script:

- "items to import" points to the `Fundamentals: aepsnorm, arevps, abvps` line.
- "imported item values" points to the `eps: F.aepsnorm`, `revs: F.arevps`, and `book: F.abvps` lines.

Below the script is a scan results table titled "Scan - 30 Items":

Date	Symbol	eps	revs	book	reported
2/28/22	AAPL	5.61	21.69	3.84	9/25/21
2/28/22	AMGN	10.28	45.34	12.00	12/31/21
2/28/22	AXP	10.02	54.23	29.14	12/31/21
2/28/22	BA	-5.07	106.00	-25.47	12/31/21
2/28/22	CAT	11.89	92.93	30.76	12/31/21
2/28/22	CRM	4.49	22.85	45.15	1/31/21
2/28/22	CSCO	2.67	11.76	9.79	7/31/21
2/28/22	CVX	8.22	81.84	72.06	12/31/21

## 17.12.160. FunBar

### Category

## Bar Data Values

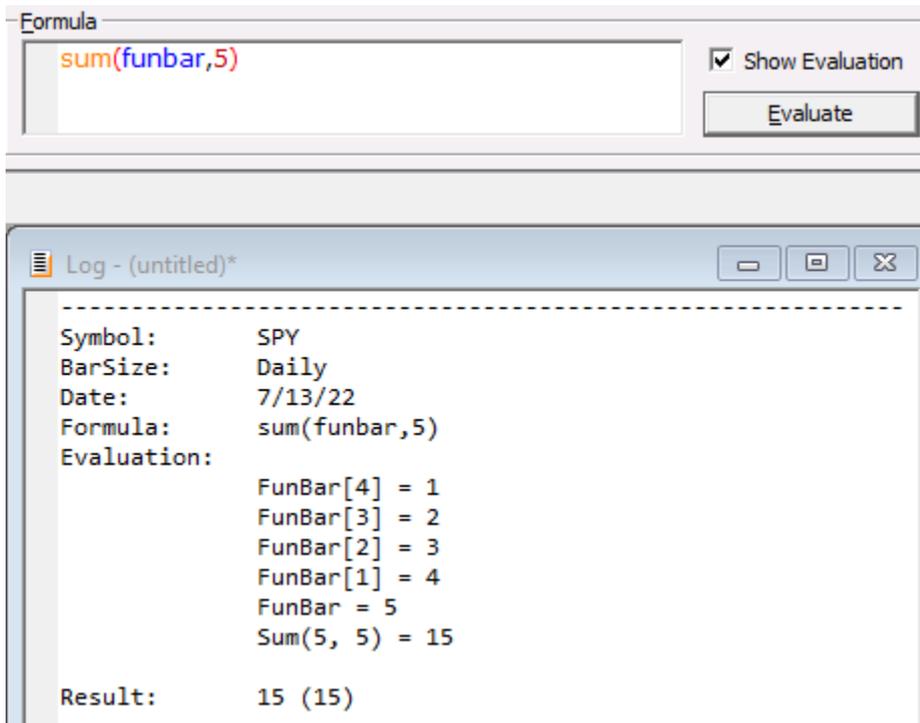
### Description

Ordinal number of a bar within the the calculation of a multi-bar function.

### Notes

*FunBar* makes it possible, in some cases, to calculate with a one-line function call a complex formula that otherwise need to be written with looping code.

As a simple example of how this works, the following shows the evaluation steps of *Sum(FunBar, 5)*:



Within the internal calculation loop of the **Sum** function, *FunBar* starts at 1 for the earliest bar, then is incremented as the calculation proceeds.

For a more complex example, see the *ehlers\_windows.rts* **example script**.

## 17.12.161. Graphs

---

### Category

#### Script Sections

### Description

Daily stats graph content definitions

### Notes

See **Graphs Section** and **Daily Stats Graph Windows**.

## 17.12.162. High or H

---

### Category

#### Bar Data Values

### Description

Current bar high price

## Notes

Either *High* or *H* can be used as the name of this value.

## 17.12.163. Highest or HHV

---

### Category

**Multi-Bar Functions**

### Description

Highest (largest) value in a series

### Syntax

Highest(expr, count) or HHV(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *Highest* or *HHV* can be used as the name of this function.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.164. HMA or HAvg

---

### Category

**Multi-Bar Functions**

### Description

Hull moving average

### Syntax

HMA(expr, count) or HAvg(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *HMA* or *HAvg* can be used as the name of this function.

The Hull moving average is calculated using nested weighted moving averages (WMA).

$HMA(expr, count)$  could be calculated as  $WMA((2 * WMA(expr, count/2) - WMA(expr, count)), SQR(count))$ .

## 17.12.165. HolidayList

---

### Category

**Settings** or **Strategy Elements**

## Description

Specifies the path to a text file which contains a list of future holidays

## Notes

If specified, the holiday list file should be a simple text file with one date per line.

There is no need to include weekend dates -- these are handled automatically -- only future market holidays need to be listed.

The purpose of the holiday list is to enable RealTest to know whether tomorrow's bar will be a new week or month when generating live orders.

Dates on which the market closes early should also be included, preceded by an asterisk, e.g. \*2022-11-25.

Early-close dates are used when generating orders that include specific times (good after time, good until date).

For backtesting, RealTest infers whether each bar is a non-trading day by the absence of data for that date.

There is therefore no reason to provide a holiday list except when **generating orders**.

A settings-level *HolidayList* will apply to all strategies in the script.

A strategy-level *HolidayList* only applies to that strategy and will override the settings-level list of both are provided.

## 17.12.166. HVOL

---

### Category

#### Indicator Functions

### Description

Historical volatility

### Syntax

HVOL(len)

### Parameters

len - lookback period

### Notes

$HVOL(len)$  could be calculated as  $StdDev(\log(c/c[1]),len) * 100 * Sqr(252)$ .

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.167. IF

---

### Category

#### General-Purpose Functions

### Description

Conditional choice function

## Syntax

IF(condition, if\_true, if\_false)

## Parameters

condition - formula specifying a true/false condition (non-zero means true)

if\_true - formula to evaluate and return the result of if condition is true

if\_false - formula to evaluate and return the result of if condition is false

## Notes

Only one the two formulas gets evaluated, depending on the condition. The function returns the result of evaluating the selected formula.

This function can also be referred to as "IIF" if you prefer.

# 17.12.168. Import

---

## Category

**Script Sections**

## Description

Data import specification

## Notes

See **Import Section**.

# 17.12.169. Include

---

## Category

**Script Sections**

## Description

Allows a script to include another script

## Syntax

*Include: path* where *path* is either a full file path (e.g. *C:\RealTest\Scripts\script.rts*) or a path relative to the Scripts folder (e.g. *Examples\script.rts*).

## Notes

A script can include any number of other scripts by using multiple include statements.

Included scripts can include other scripts, and so on (it's recursive).

RealTest ensures that each specific script is included only once.

Included scripts must contain entire script section definitions -- they cannot, for example, include just one part of a Strategy definition such as a set of exit-related items.

The combined scripts are parsed as one large script, so each item name (Data items etc.) must be unique across the combined scripts.

Suggested names in the script editor (auto-complete) will include any names defined in included scripts.

*Include* is applied in all script **run modes**. To include another script in a specific mode only, use **OrdersInclude**, **ScanInclude** or **TestInclude**.

## 17.12.170. IncludeList

---

### Category

#### Import Specification

### Description

List of one or more symbols to include when **importing data**

### Notes

An **Import Section** can have any number of *IncludeList* statements.

Each *IncludeList* statement can take one of the following forms:

- One or more symbols, separated by commas (up to a maximum of 260 characters in total)
- A path to a TXT file containing a list of symbols (one symbol per line with no commas)
- A path to a CSV file in which a column of symbols has a name that contains "symbol" or "ticker" or "underlying"
- The name of a **Norgate** watchlist, preceded by a dot (Norgate only)

The symbols from all of the specified include lists are combined into a single internal list.

If there are duplicates, only one copy of each symbol will be imported.

A special syntax, *SYMBOL>ALIAS*, can optionally be used to rename a symbol after it is imported (e.g. AAPL>APPLE). This is meant to be used when you actually want to import the same symbol from two different sources, so that each can be uniquely identified.

The number of the list in which a symbol first appears (whether literally or via an external file) is stored with the symbol and becomes the value return by **ListNum** when that symbol is the current context.

Though a symbol is only imported once, it can be listed in more than one *IncludeList*. The **InList(n)** function can be used to check whether the current symbol was among the symbols in the *n*th *IncludeList*.

An *IncludeList* can optionally be given a name. This is done by adding a squiggly-brace comment containing the name in quotes after the list definition, like this:

```
▼ Import:
  DataSource: Norgate
  IncludeList: SPY, GLD, TLT {"asset_class"}
```

When a data file was imported using named include lists, a list name can optionally be used in place of the list number in the *InList* function, like this:

```
EntrySetup: EndOfWeek and InList("asset_class")
```

This can be a useful technique when combining multiple strategies in one script and therefore one *Import* definition.

By default any date for which at least one imported symbol has a bar becomes part of the global date list used to run tests and scans.

The symbols from an *IncludeList* can optionally be excluded from the global date list. This is useful, for example, when importing special symbols for risk-free interest rates, currency conversion etc. Often these symbols will include bars for dates which are otherwise market holidays.

To prevent the symbols in an *IncludeList* from being added to the global date list, add the comment `{//}` to the same line in the script.

If the list also requires a list name, use the same special comment, e.g. `{"currency"//}`.

## 17.12.171. InXXX

---

### Category

**Bar Data Values**

### Description

**Norgate** Index constituency lookup (if specified during import)

### Syntax

*InXXX* where 'XXX' is a specific index symbol, e.g. *InSPX*

### Notes

For *InXXX* to return anything other than 0, the corresponding **Constituency** data series must have been included in the **Import**.

See the *Constituency* topic for all the details about how this works.

As with any other bar data value, you can use offset syntax with these.

For example, *InSPX > InSPX[1]* would be a way to find dates on which stocks were added to the S&P 500, and *InSPX < InSPX[1]* would find dates on which they were removed.

## 17.12.172. InfoID

---

### Category

**Stock/Contract Information**

### Description

Norgate Asset ID for this symbol

### Notes

Automatically provided when data is imported from **Norgate**.

To provide this field in your own **SymInfoFile**, use the *AssetId* column.

The Asset ID is an integer.

## 17.12.173. InfoDelist / InfoExpiry

---

### Category

**Stock/Contract Information**

### Description

Stock delisting date or futures contract expiration date

### Notes

Automatically provided when data is imported from **Norgate**.

To provide this field in your own **SymInfoFile**, use the *DelistDate* column.

The format is YYYYMMDD.

## 17.12.174. InfoFloat

---

### Category

**Stock/Contract Information**

### Description

The number of shares in circulation (float) for a stock

### Notes

This piece of information is provided automatically when data is imported from **Norgate**.

This is not a historical data series. The number returned is always the current value.

## 17.12.175. InfoGICS

---

### Category

**Stock/Contract Information**

### Description

Global Industry Classification Standard code

### Notes

Automatically provided when data is imported from **Norgate**.

See the **GICS** entry in Wikipedia for details.

The *InfoGICS* code is available regardless of the **Classification** scheme used during import.

To isolate the upper digits of the code, use the **Top** function.

If the import classification scheme was *GICS* then level-specific names such as **?Sector** are also available.

## 17.12.176. InfoMargin

---

### Category

**Stock/Contract Information**

### Description

Futures contract margin requirement

### Notes

Automatically provided when data is imported from **Norgate**.

To provide this field in your own **SymInfoFile**, use the *TRBC* column.

Note that this is just a current value, not a historical data series.

## 17.12.177. InfoShares

---

### Category

## Stock/Contract Information

### Description

Number of shares in existence (outstanding) for a stock

### Notes

This piece of information is provided automatically when data is imported from **Norgate**.

This is not a historical data series. The number returned is always the current value.

## 17.12.178. InfoTRBC

---

### Category

#### Stock/Contract Information

### Description

Thomson Reuters Business Classification code

### Notes

Automatically provided when data is imported from **Norgate**.

See the **TRBC** entry in Wikipedia for details.

The *InfoTRBC* code is available regardless of the **Classification** scheme used during import.

To isolate the upper digits of the code, use the **Top** function.

If the import classification scheme was *TRBC* then level-specific names such as **?Sector** are also available.

## 17.12.179. InList

---

### Category

#### Stock/Contract Information

### Description

Checks whether this symbol was part of a specific *IncludeList* when it was imported

### Parameters

The single parameter to this function can be either a list number, e.g. *InList(2)* or a list name, e.g. *InList("my\_list")*

### Notes

If the current data file was imported using multiple *IncludeList* statements, *InList* can be used to filter symbols based on their include list membership.

This is especially useful when a strategy uses one set of symbols for trading and a different set of symbols for calculating an indicator.

Here is a contrived example:

```

▽ Import:
  DataSource: Norgate
  IncludeList: .NASDAQ 100
  IncludeList: .Dow Jones Industrial Average
  // etc.

▽ Data:
  NasUp: #sum InList(1) and C > MA(C,20)

▽ Strategy: test
  EntrySetup: InList(2) and NasUp > 80 and // whatever

```

A breadth indicator is constructed using NDX components by counting how many of them are above their 20-day averages.

The strategy trades only DJIA components and only when the NDX breadth indicator is above 80.

This example could also have used *Constituency*:  $\$NDX$ ,  $\$DJI$  and replaced the *InList(1)* with *InNDX* and *InList(2)* with *InDJI*. Using historical index **constituency** (requires Norgate Platinum subscription) is recommended for backtests in order to avoid survivorship bias. The *InList* technique is shown here is only advisable for daily scans or very recent backtests.

For examples of other ways to use *InList*, see *combined.rts* and *vigilant\_asset\_allocation.rts* in the *Examples* folder.

See the example at the end of the **IncludeList** page for information on how to refer to an include list by name rather than by number.

See also **ListNum**, which returns the number of the **first** list that included a symbol.

## 17.12.180. IsExit

---

### Category

**Current Position Information**

### Description

True when the formula being evaluated pertains to a position being exited

### Notes

*IsExit* is provided for the few strategy formulas that are evaluated at both entry time and exit time.

This include:

- **Commission**
- **Slippage**
- **OrderNote / OrderExtra**

Use *IsExit* in any of the above formulas as needed to differentiate entry vs. exit values.

For example in a long strategy you could account for the extra SEC Section 31 fee on the sale of shares:

```
Commission: 0.005 * Shares + if(IsExit, 8 * FillValue/1e6, 0)
```

## 17.12.181. IsNan

---

### Category

**General-Purpose Functions**

### Description

Returns 1 (true) if formula can't be evaluated, or 0 (false) if it can.

## Syntax

IsNan(value)

## Parameters

value - formula

## Notes

The most common reason that a formula can't be evaluated is that it tries to refer to more bars of data than are currently available. For example, if you import data for SPY starting 1/2/2010 and then try to calculate a 200-day moving average for 2/1/2010, there are not enough bars yet, so `Avg(C,200)` will return NAN, and `IsNan(AVG(C,200))` will return TRUE.

Once any term of any formula evaluates to NAN, the result of the entire formula becomes NAN. This is why it can be useful to find out if a specific term is NAN before evaluating the entire formula. The **IIF** function is useful for this purpose.

The constant NAN can also be used in any formula (wherever a number is expected) to force the formula evaluation to fail. (NAN stands for "not a number".)

A great use for this NAN constant is when calculating a **breadth function** where you only want **index constituents** to be included for each date.

Say you want to know the **median** price per share of all stocks in the S&P 500 for each date in the past.

This example shows how to specify the **Import**, set up the **Data** section and run the **Scan**:

```
Active Script - C:\REALTEST\Examples\index_breadth.rts*
Import:
  DataSource: Norgate
  IncludeList: .S&P 500 Current & Past
  IncludeList: SPY
  IndexList: $SPX
  StartDate: 1/2/10
  EndDate: Latest
  SaveAs: spx.rtd
Data:
  MedC: #Median C
  MedC0: #Median iif(index(1), C, 0)
  MedCX: #Median iif(index(1), C, nan)
Scan:
  // only show each day's value once
  Filter: Symbol = $SPY
  MedC: MedC
  MedC0: MedC0
  MedCX: MedCX
```

Date	Symbol	MedC	MedC0	MedCX
6/30/20	SPY	60.61	57.29	79.64
6/29/20	SPY	59.57	56.76	78.40
6/26/20	SPY	58.38	55.79	77.59
6/25/20	SPY	59.81	56.40	78.98
6/24/20	SPY	59.30	55.74	78.07
6/23/20	SPY	61.27	57.82	81.10
6/22/20	SPY	61.42	57.56	80.95
6/19/20	SPY	60.95	56.06	79.55
6/18/20	SPY	61.76	56.99	78.81
6/17/20	SPY	62.06	56.93	79.17

Notice how each of the three columns shows different values.

*MedC* ignores index constituency so it includes all symbols that are no longer in the index (mostly

penny stocks now).

*MedCO* filters out non-constituents, but by substituting 0 for their prices, it skews the median value even lower than the the prior column.

*MedX* returns the correct values, because when a NAN is encountered by the cross-sectional calculator, that item is simply excluded from the calculation.

## 17.12.182. IsOrder

---

### Category

**Current Position Information**

### Description

Returns 1 (true) if the current stock passed **EntrySetup** for the specified strategy and was not "skipped" for any reason (**MaxSetups**, **MaxPositions**, etc.)

### Notes

Without **Combined** or **Extern**, *IsOrder* applies only to the current stock in the current strategy.

When used with *Combined* or *Extern*, *IsOrder* returns the count of strategies for which the current stock is a setup and has not been "skipped".

## 17.12.183. IsSetup

---

### Category

**Current Position Information**

### Description

Returns 1 (true) if the current stock passed **EntrySetup** for the specified strategy

### Notes

Without **Combined** or **Extern**, *IsSetup* will by definition be 0 (false) when referenced in *EntrySetup*, or 1 (true) in any other entry-related formula, since the others are only evaluated for setups.

When used with *Combined* or *Extern*, *IsSetup* returns the count of strategies for which the current stock met the *EntrySetup* condition.

Since *IsSetup* remains true even for skipped setups, **IsOrder** is generally more useful for avoiding duplicate entries in the same symbol.

## 17.12.184. Item

---

### Category

**General-Purpose Functions**

### Description

Reference a Data Item, Library Item or Test Parameter by name

## Syntax

Item(name, ...)

## Parameters

name - string specifying an item name

## Notes

This function works the same way **Format** does in terms of its parameters. The output is then treated as an item name.

For example, `Item("entry_rule_{#}", rule_num)` would look for an item called "entry\_rule\_1" if rule\_num had a value of 1.

The referenced item must be the name of an item in either the **Data** or **Library** section of the current script (or an included script).

One use of this feature is to have a set of different factors that could be parts of an **EntrySetup** or **ExitRule**, for which you can run a kind of optimization that tests each factor separately.

Here is a template for how this might work:

```
▼ Parameters:
  entry_cond_num: from 1 to 3
  exit_cond_num: from 1 to 3

▼ Library:
  entry_cond1: (formula)
  entry_cond2: (formula)
  entry_cond3: (formula)
  exit_cond1: (formula)
  exit_cond2: (formula)
  exit_cond3: (formula)

▼ Strategy: test
  EntrySetup: item("entry_cond{#}", entry_cond_num)
  ExitRule: item("exit_cond{#}", exit_cond_num)
```

(The "(formula)" items are just placeholders for meaningful trading conditions.)

## 17.12.185. KAMA

---

### Category

#### Multi-Bar Functions

### Description

Kaufman Adaptive Moving Average

### Syntax

KAMA(expr, erlen, fastest, slowest)

### Parameters

expr - data series formula

erlen - lookback length for the Efficiency Ratio calculation used to adapt the EMA length

fastest - shortest EMA equivalent lookback period

slowest - longest EMA equivalent lookback period

### Notes

KAMA is a kind of exponential moving average which adapts its weighting factor (often mistaken for a lookback length) at each bar of its calculation by computing the Kaufman Efficiency Ratio and

using that to determine that bar's weighting.

The most common parameters are  $KAMA(expr, 10, 2, 30)$ .

See [stockcharts.com](https://stockcharts.com) for further details.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with non-variable length parameters.

## 17.12.186. KBBOT

---

### CategoryCategory

#### Indicator Functions

### Description

Keltner band (channel) bottom

### Syntax

$KBBOT(len, atrs)$

### Parameters

len - lookback period

atrs - number of ATRs

### Notes

It's not clear that there's a "standard" way to implement Keltner channels. The center line could be either a simple or an exponential average. The ATR might use a simple average, EMA, or Wilder's smoothing. This function is a shortcut for  $EMA(C, len) - atrs * EMA(TR, len)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.187. KBTOP

---

### Category

#### Indicator Functions

### Description

Keltner band (channel) top

### Syntax

$KBTOP(len, atrs)$

### Parameters

len - lookback period

atrs - number of ATRs

### Notes

It's not clear that there's a "standard" way to implement Keltner channels. The center line could be either a simple or an exponential average. The ATR might use a simple average, EMA, or Wilder's smoothing. This function is a shortcut for  $EMA(C, len) + atrs * EMA(TR, len)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.188. KeepAdjusted

---

### Category

#### Import Specification

### Description

Whether to keep imported price and volume data values split-adjusted rather than as-traded

### Choices

*False* - don't keep adjusted (default)

*True* - keep adjusted

### Notes

In most cases, *False* is the recommended choice.

If *True* is selected, all prices used or shown in any context are adjusted for future splits.

See **Split Handling** for details on how RealTest uses unadjusted data with knowledge of split factors to provide realistic as-traded backtests.

## 17.12.189. KeepRedundant

---

### Category

#### Import Specification

### Description

Whether to keep multiple redundant stocks of the same company in the data file

### Choices

*False* - don't keep redundant stocks (default)

*True* - keep redundant stocks

### Notes

By default, when you import data that includes company names, RealTest uses the names to look for redundancies.

Specifically, if two names include " class " or " series " (note the spaces on either side of each string), and the strings up to and including that word are identical, then the two stocks are considered redundant.

When a pair of redundant stocks is found, the most recent 20-day average turnover is calculated for each, and only the one with the larger average turnover is kept.

Redundant stocks that were removed are shown in the Import Log if one was created, like this example for the Norgate *Nasdaq 100 Current & Past* watchlist:

```
Stocks removed because another class of the same stock has higher average turnover:  
BATRA (Liberty Media Braves Series A Common)  
CHTRQ-200911 (Charter Communications Inc Class A Common)  
CMCSK-201512 (Comcast Corp Class A Special Non-Voting Common)  
DISCK (Discovery Inc Series C Common)  
FOX (Fox Corp Class B Common)  
GOOG (Alphabet Inc Class C Common)  
LBTYA (Liberty Global PLC Class A Common)  
LILA (Liberty Latin America Ltd Class A Common)  
LMCK-201604 (Liberty Media Corp Series C Common)  
TFCF-201903 (Twenty-First Century Fox Inc Class B Common)
```

## 17.12.190. KeepTrades

---

### Category

#### Settings

### Description

Specifies which categories of trades to store in each test results record

### Choices (multiple, separated by commas)

*None* - don't keep any trades (a good choice when running large optimizations)

*Strategy* - keep regular strategy trades (the most common choice)

*Benchmark* - keep benchmark strategy trades

*Skipped* - keep trades (setups) that were skipped for various reasons

*All* - a one-word shortcut for all of the above (except none)

### Notes

If *KeepTrades* is not specified in a script then the choices from the **Settings Panel** will remain unchanged and be used.

## 17.12.191. Kurtosis

---

### Category

#### Multi-Bar Functions

### Description

Statistical measure of the heaviness of the tail of a distribution of values

### Syntax

Kurtosis(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

*Kurtosis* is calculated in the way that Excel would calculate a KURT.P function if it had one, i.e., as if the set of *count* values is the entire population.

The specific formula used is shown below, in the "kurt" item:

#### ▼ Data:

```
expr: roc(c,1)
mean: avg(expr, count)
sdev: sqr(sum((expr - this(mean)) ^ 2, count) / count)
skew: (1 / count) * (sum((expr - this(mean)) ^ 3, count) / sdev ^ 3)
kurt: (1 / count) * (sum((expr - this(mean)) ^ 4, count) / sdev ^ 4) - 3
```

This also illustrates how these statistical functions could be calculated in the **Data Section** of a script, though since they're provided built-in, there's no reason to do so.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **StdDev** and **Skewness**.

## 17.12.192. Left

---

### Category

**String Functions**

### Description

Return the left end of a string

### Syntax

Left(string, length)

### Parameters

string - a **literal string** or **string function** result

length - the number of characters to include

## 17.12.193. LegacyMode

---

### Category

**Settings**

### Description

Process setups and apply constraints as was done prior to the introduction of top-down mode in release 2.0.26

### Choices

*True* - use legacy mode

*False* - use top-down mode (default)

### Notes

In *Legacy* mode, there was no support for **Combined** or **StatsGroup** constraints. Each strategy had to calculate its individual constraints with explicit references to other strategy's current investment levels to model any kind of multi-strategy caps.

Also in *Legacy* mode, position sizes for strategy constraint checks were based on **FillPrice**, implying that share **Quantity** could be calculated at the moment of entry.

In the newer top-down mode, constraints are checked at multiple levels -- *Strategy*, *StatsGroup* and *Combined* -- and are based on the **OrderPrice** of each setup

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.194. Length

---

### Category

**String Functions**

### Description

Return the number of characters in a string

### Syntax

Length(string)

### Parameters

string - a **literal string** or **string function** result

## 17.12.195. Library

---

### Category

**Script Sections**

### Description

Named formulas calculated when they are referenced, using the current context

### Notes

See **LibrarySection** for a more detailed description.

## 17.12.196. LimitExtra

---

### Category

**Strategy Elements**

### Description

Excursion beyond a limit price to require before assuming the limit order was filled

### Input

Any formula specifying dollars per share (points)

### Notes

If this element is not specified or evaluates to 0, then limit orders are assumed to fill completely whenever the limit price is within the range of the day. This will be true even if the price happens to match the exact high or low.

In live trading it is not realistic to assume that limit orders very near the high or low of the day will be filled.

By using *LimitExtra*, you can make your trading model more realistic.

The value returned by this formula should specify a number of points (price per share) to require as the minimum extra excursion beyond the limit price.

A very simple assumption is *LimitExtra: TickSize*

For stocks, this is equivalent to *Limitextra: 0.01*

To assume that a larger spread is required for higher priced stocks, use a percentage instead, such as *LimitExtra: 0.001 \* C {0.1% of price}*

It makes sense to use *LimitExtra* if you use standard limit orders when trading your strategy, or use **LimitSlip** if you use "market if touched" or "not held" (discretionary) limit orders.

See also **EntryLimit**, **ExitLimit**.

## 17.12.197. LimitSlip

---

### Category

**Strategy Elements**

### Description

Slippage amount, in points (dollars per share or contract), for each **limit order** transaction

### Input

Any formula specifying dollars per share or contract (points)

### Notes

Defines the amount of slippage to apply to each limit order transaction, in price points.

*LimitSlip* is applied to any transaction that occurs at an **EntryLimit** or **ExitLimit** price.

If *LimitSlip* is not specified then **Slippage** is applied instead.

When a strategy uses both **EntryStop** and **EntryLimit** (enters positions with a stop-limit order) and the fill is at the stop price, **StopSlip** is applied.

When using *LimitSlip: 0* consider testing with non-zero **LimitExtra**.

## 17.12.198. LinReg

---

### Category

#### Multi-Bar Functions

### Description

Linear regression

### Syntax

LinReg(expr, {expr2,} count)

### Parameters

expr - data series formula (Y values)

expr2 - optional second data series formula (X values -- a linear series from 1 to *count* is used if omitted)

count - lookback period

### Notes

Calculates the endpoint of a linear regression of *expr* evaluated for the previous *count* bars.

This is equivalent to **YInt** + (**Slope** \* latest\_x\_value).

This function is equivalent to =FORECAST in Excel.

## 17.12.199. ListNum

---

### Category

#### Stock/Contract Information

### Description

The number of the first import *IncludeList* which contained this symbol

### Notes

See also **InList**, which can be used to check whether the current symbol was in a specific **IncludeList**.

For an import where each symbol only appears in one include list, *ListNum==n* is equivalent to *InList(n)*.

*InList(n)* will always be true when *ListNum==n*, but *ListNum==n* will only be true if *n* is the first list

to contain the symbol.

When **CIIFamily** and **CIILevel** are used with **Norgate Import**, corresponding industry index symbols are automatically imported for each specific symbol that has one. These automatically imported symbols are placed in a virtual include list with the number **99**.

See *industry\_indices.rts* in the *Examples* folder for further clarification on how this CII mechanism works.

## 17.12.200. Log

---

### Category

#### General-Purpose Functions

### Description

Natural logarithm of a number

### Syntax

Log(value)

### Parameters

value - formula

### Notes

A daily log return could be calculated as  $\text{Log}(S.\text{Equity}/S.\text{Equity}[1])$ .

Weekly would be  $\text{Log}(S.\text{Equity}/S.\text{Equity}[5])$ .

The log return of a trade (for display in the trade list window, assuming long side) would be  $\text{Log}(T.\text{PriceOut}/T.\text{PriceIn})$ .

## 17.12.201. LogFile

---

### Category

#### Import Specification

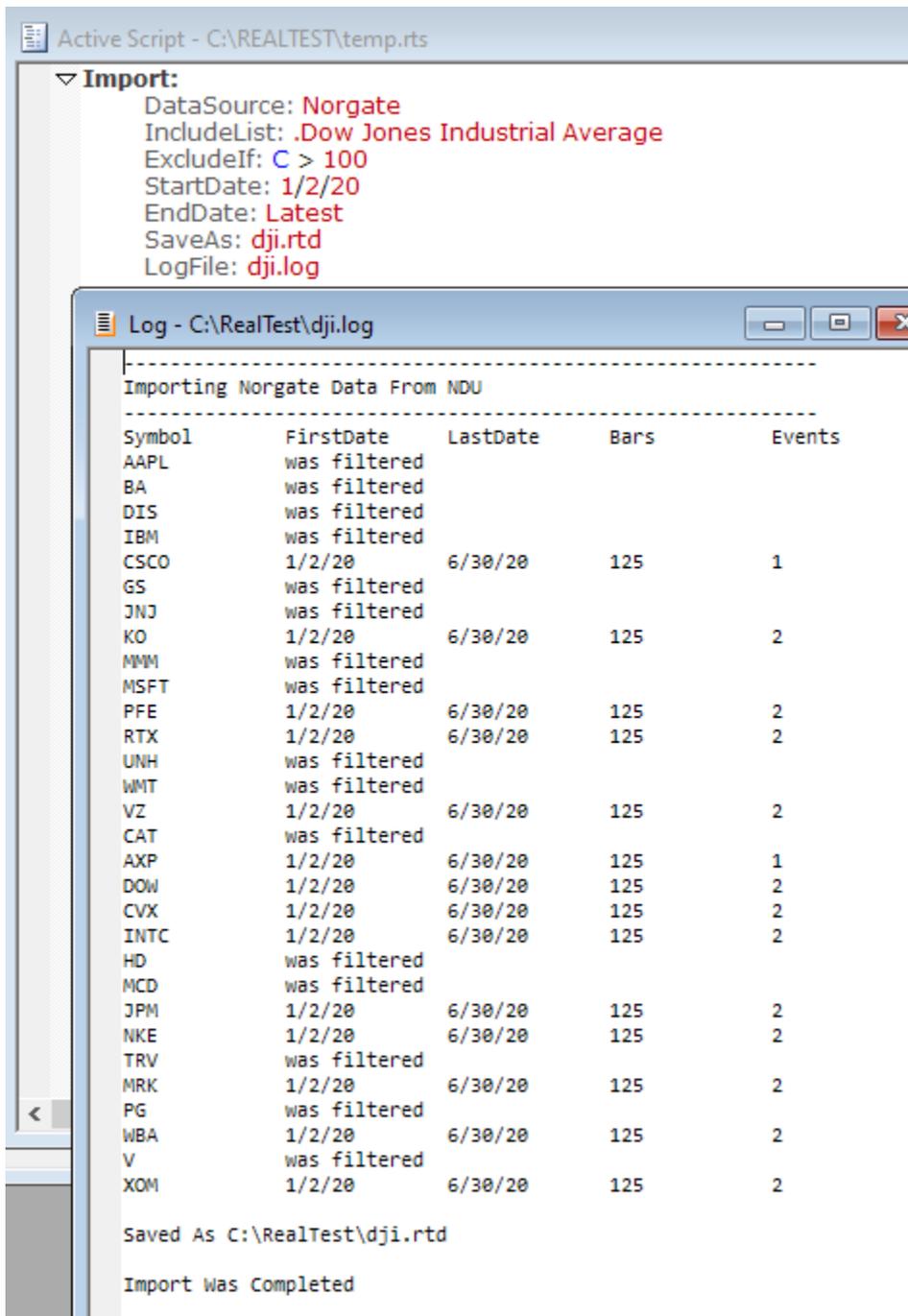
### Description

Path and name of an import log file to create

Notes

If *LogFile* is specified, the file is created at the start of the import, written to during the import, and then opened to a **Log Window** when the import finishes.

Here is a simple example where the DJIA components were imported but stocks above \$100/share were excluded from the import. The log shows which those were.



The "Events" column in this case shows how many ex-dividend dates there were in this time period for each stock. If a custom **event list** is used, the count of those events would be shown here as well.

## 17.12.202. Low or L

### Category

#### Bar Data Values

### Description

Current bar low price

### Notes

Either *Low* or *L* can be used as the name of this value.

## 17.12.203. Lowest or LLV

---

### Category

#### Multi-Bar Functions

### Description

Lowest (smallest) value in a series

### Syntax

Lowest(expr, count) or LLV(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *Lowest* or *LLV* can be used as the name of this function.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.204. MA or Avg

---

### Category

#### Multi-Bar Functions

### Description

Simple Moving Average

### Syntax

MA(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *MA* or *Avg* can be used as the name of this function.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.205. MACD

---

### Category

#### Indicator Functions

### Description

Moving Average Convergence Divergence

### Syntax

MACD(len1, len2)

### Parameters

len1 - faster moving average period (typically 13)

len2 - slower moving average period (typically 26)

### Notes

MACD is the difference between two moving averages.

It is calculated as  $EMA(C, len1) - EMA(C, len2)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with non-variable lengths.

## 17.12.206. MACDH

---

### Category

#### Indicator Functions

### Description

MACD Histogram

### Syntax

MACDH(len1, len2, len3)

### Parameters

len1 - faster moving average period (typically 13)

len2 - slower moving average period (typically 26)

len3 - signal smoothing period (typically 9)

### Notes

MACDH is the difference between **MACD** and its signal line.

It is calculated as  $MACD(len1, len2) - MACDS(len1, len2, len3)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with non-variable lengths.

## 17.12.207. MACDS

---

### Category

#### Indicator Functions

### Description

MACD Signal Line

### Syntax

MACDS(len1, len2, len3)

### Parameters

len1 - faster moving average period (typically 13)

len2 - slower moving average period (typically 26)

len3 - signal smoothing period (typically 9)

## Notes

MACDS is an EMA-smoothed version of **MACD**.

This is also known as the "signal line".

It is calculated as  $EMA(MACD(len1, len2), len3)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with non-variable lengths.

## 17.12.208. MarginIntPct

---

### Category

**Settings**

### Description

Interest rate paid for negative daily excess cash (margin loan)

### Notes

If the settings also provide a **RiskFreeRateSym** then *MarginIntPct* is **added** to the current daily interest rate as determined by today's value of the risk-free rate series. In this case, *MarginIntPct* would typically be positive, e.g. 1.5 if your broker charges 1.5% more than the fed funds rate for margin loans.

If *RiskFreeRateSym* is provided but *MarginIntPct* is not provided or is 0 then no margin interest is charged. You must specify a non-zero value of *MarginIntPct* to include margin interest payments in your backtests.

If *MarginIntPct* is provided when there is no *RiskFreeRateSym* then *MarginIntPct* is simply a fixed annual interest rate. In this case the rate should be expressed as a negative number, e.g. -5 to pay margin interest at an annualized rate of 5%.

Daily net interest received or paid is added to combined **S.Equity** and therefore becomes part of the total return of the backtest.

The stats series **S.Interest** can be used to graph or reference the cumulative net interest received or paid in the account.

See also **CashIntPct** which specifies the rate received for positive excess cash.

## 17.12.209. MarkToMarket

---

### Category

**Strategy Elements**

### Description

Determines whether strategy equity includes open-position mark-to-market value

### Choices

*True* - equity includes mark-to-market (default)

*False* - equity is closed-trade-basis

### Notes

This setting changes the value of **S.Equity**. By default, *S.Equity* is updated daily as (starting capital) + (net closed trade profits) + (net open position mark to market values). If *MarkToMarket: False* is added to a strategy, then its *S.Equity* series becomes (starting capital) + (net closed trade profits).

This setting also affects the internal calculations of **S.DDDIr**, **S.DDPct**, **S.MaxDDDir**, **S.MaxDDPct**, **S.NetDir** and **S.NetPct**. It also affects the default value of **Allocation** if no formula is provided,

since that default is *Combined(S.Equity)*.

## 17.12.210. Match

---

### Category

#### String Functions

### Description

Determine whether a string matches a pattern

### Syntax

```
Match(string, pattern)
```

### Parameters

string - a **literal string** enclosed in either double or single quotes ("string" or 'string') or a **string function**

pattern - a string (or function) defining the pattern to check the string against

### Notes

The string and pattern are compared character by character.

? in the pattern can match any single character in the string.

\* in the pattern can match zero or more characters in the string.

Otherwise, the characters must match exactly (ignoring case for letters).

### Examples

```
Match("test1", "test2") is False
```

```
Match("test1", "test?") is True
```

```
Match("test1", "*test") is False
```

```
Match("test1", "test*") is True
```

### Common Uses

Does *string* contain "XYZ"?

➤ `Match(string, "*XYZ*")`

Does a *string* start with "XYZ"?

➤ `Match(string, "XYZ*")`

Does *string* end with "XYZ"?

➤ `Match(string, "*XYZ")`

### Symbol Information (Norgate)

Was a stock delisted in 1995?

➤ `Match(?Symbol, "*-1995??")`

Is the current symbol an individual ES futures contract?

➤ `Match(?Symbol, "ES-*")`

Does the industry of the current stock have anything to do with oil?

➤ `Match(?Industry, "*oil*")`

## 17.12.211. Max

---

### Category

#### General-Purpose Functions

### Description

Largest of a group of values

### Syntax

Max(value1, value2, ...)

### Parameters

value1 - any formula

value2 - any formula

... - any number of additional comma-separated formulas

### Notes

This function evaluates each of the formulas passed to it and returns the largest value found.

## 17.12.212. MaxN

---

### Category

#### General-Purpose Functions

### Description

Nth largest of a group of values

### Syntax

MaxN(N, value1, value2, ...)

### Parameters

N - number from 1 to count of values

value1 - any formula

value2 - any formula

... - any number of additional comma-separated formulas

### Notes

This function evaluates each of the formulas passed to it and returns the Nth largest value found.

*MaxN*(1, value1, value2) is the same as **Max**(value1, value2).

## 17.12.213. MaxEntries

---

### Category

#### Strategy Elements

### Description

Caps the number of actual position entries per day

### Input

Any formula that returns a count

## Notes

*MaxEntries* is a **LegacyMode** formula and is not compatible with in-advance order placement.

Use **MaxNewPos** in the default mode if you want to cap the daily new position count at order time.

---

*The following notes apply only to legacy mode.*

The number of positions that a strategy can enter per day is determined by evaluating the **MaxSetups**, *MaxEntries*, **MaxExposure**, **MaxInvested**, and **MaxPositions** formulas at entry time.

If **EntryScore** is not specified, setups will be entered in alphabetical order by symbol.

For strategies that enter all setups at market, or to cap the number of limit or stop orders placed to the number you would actually want to be filled (without look-ahead bias), it is recommended to use top-down mode.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

## 17.12.214. MaxExposure

---

### Category

**Strategy Elements**

### Description

Open position exposure limit for a strategy or group of strategies

### Input

Any formula specifying a percentage

### Notes

*MaxExposure* is one of the **Capacity Constraints** formulas used in the setup selection process.

The *Exposure* of a position is 100 times the initial dollar value (cost) of the position (based on order price) divided by the strategy's **S.Alloc** value.

A setup is only selected if its exposure, when added to those of currently open positions and previously selected setups, would not exceed the value returned by this formula.

If a setup cannot be selected due to this constraint and the strategy specifies **Reduce: True**, then the position's **Quantity** will be reduced to allow the position to be ordered for entry at smaller size if possible.

Setups skipped for this reason display "max exposure" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxExposure* is not specified then there is no limit placed on the percent exposure, though there might still be investment limits if the strategy specifies **MaxInvested** and/or **MaxPositions**.

To cap exposure at 100% (e.g. 20 positions at 5% each), use *MaxExposure: 100*, and so on.

Note that opening gaps beyond a specified order price may cause actual exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended exposure".

See Also: **Backtest Engine Details** and **Capacity Constraints**

---

When using **Legacy Mode**:

- *MaxExposure* is checked at position entry time rather than setup selection time
- **EntryScore** is used to determine which entries got priority
- *MaxExposure* cannot be used in **Combined** or **StatsGroup**

## 17.12.215. MaxInvested

---

### Category

#### Strategy Elements

### Description

Open position investment limit for a strategy or group of strategies

### Input

Any formula specifying a dollar amount

### Notes

*MaxInvested* is one of the **Capacity Constraints** formulas used in the setup selection process.

The *Investment* of a position is its share quantity times its entry price.

A setup is only selected if its investment value, when added to those of currently open positions and previously selected setups, would not exceed the value returned by this formula.

If a setup cannot be selected due to this constraint and the strategy specifies **Reduce: True**, then the position's **Quantity** will be reduced to allow the position to be ordered for entry at smaller size if possible.

Setups skipped for this reason display "max invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxInvested* is not specified then there is no limit placed on the dollar investment level, though there might still be investment limits if the strategy specifies **MaxExposure** and/or **MaxPositions**.

To cap investment at the strategy's current allocation, use *MaxInvested: S.Alloc*.

See Also: **Backtest Engine Details** and **Capacity Constraints**

---

When using **Legacy Mode**:

- *MaxInvested* is checked at position entry time rather than setup selection time
- **EntryScore** is used to determine which entries got priority
- *MaxInvested* cannot be used in **Combined** or **StatsGroup**

## 17.12.216. MaxNewExp

---

### Category

#### Strategy Elements

### Description

Caps the added exposure from entry orders placed per day by a strategy

### Input

Any formula that returns a percentage

### Notes

At each step of the setup selection process, *MaxNewExp* is considered along with each of the other top-down constraints.

If, for example, *MaxNewExp* is 20 and there have already been two other setups with 10% position size selected for order placement, this setup will be rejected even if adding it would not violate any other constraint.

See Also: [Backtest Engine Details](#) and [Capacity Constraints](#)

---

When using **Legacy Mode**:

- *MaxNewExp* is not supported

## 17.12.217. MaxNewInv

---

### Category

**Strategy Elements**

### Description

Caps the added investment from entry orders placed per day by a strategy

### Input

Any formula that returns a dollar amount

### Notes

At each step of the setup selection process, *MaxNewInv* is considered along with each of the other top-down constraints.

If, for example, *MaxNewInv* is \$10,000 and there have already been two other setups with \$5,000 position size selected for order placement, this setup will be rejected even if adding it would not violate any other constraint.

See Also: [Backtest Engine Details](#) and [Capacity Constraints](#)

---

When using **Legacy Mode**:

- *MaxNewInv* is not supported

## 17.12.218. MaxNewPos

---

### Category

**Strategy Elements**

### Description

Caps the number of entry orders placed per day when a strategy has more setups than can be entered

### Input

Any formula that returns a count

### Notes

At each step of the setup selection process, *MaxNewPos* is considered along with each of the other top-down constraints.

If, for example, *MaxNewPos* is 3 and there have already been three other setups selected for order placement, this setup will be rejected even if adding it would not violate any other constraint.

See Also: [Backtest Engine Details](#) and [Capacity Constraints](#)

---

When using **Legacy Mode**:

- *MaxNewPos* is not supported

## 17.12.219. MaxPerTurn

---

## Category

### Strategy Elements

## Description

Defines how many setups per turn a strategy can add during the top-down setup selection process

## Input

Any formula that returns a count

## Notes

The default is 1, meaning that each strategy selects only one setup per turn of the selection process.

To allow a strategy to select all of its setups on its first turn, use a large number (e.g. its maximum positions).

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.220. MaxPositions

---

## Category

### Strategy Elements

## Description

Open position count limit for a strategy or group of strategies

## Input

Any formula specifying a count

## Notes

*MaxPositions* is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if the count of currently open positions plus previously selected setups plus one would not exceed the value returned by this formula.

Setups skipped for this reason display "max positions" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxPositions* is not specified then there is no limit placed on the number of open positions, though there might still be investment limits if the strategy specifies **MaxExposure** and/or **MaxInvested**.

See Also: **Backtest Engine Details** and **Capacity Constraints**

---

When using **Legacy Mode**:

- *MaxPositons* is checked at position entry time rather than setup selection time
- **EntryScore** is used to determine which entries got priority
- *MaxPositions* cannot be used in **Combined** or **StatsGroup**

## 17.12.221. MaxSameCat

---

## Category

### Strategy Elements

## Description

Limits the number of same-category positions that can be open simultaneously in a strategy or group of strategies

### Input

Any formula specifying a count

### Notes

*MaxSameCat* is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if the count *with the same category as this setup* of currently open positions plus previously selected setups plus one would not exceed the value returned by this formula.

Setups skipped for this reason display "max same cat" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxSameCat* is not specified then there is no limit placed on the number of same-category positions.

See Also: **Backtest Engine Details** and **Capacity Constraints**

---

When using **Legacy Mode**:

- *MaxSameCat* is checked at position entry time rather than setup selection time
- **EntryScore** is used to determine which entries got priority
- *MaxSameCat* cannot be used in **Combined** or **StatsGroup**

## 17.12.222. MaxSameSym

---

### Category

#### Strategy Elements

### Description

Specifies how many positions in the same symbol can be open at the same time in a strategy or group of strategies

### Input

Any formula specifying a count (if not specified, the default is 1 for a strategy, unlimited for a group)

### Notes

Specifying a value greater than one for *MaxSameSym* replaces the former *Pyramid: True* strategy setting.

In the default top-down mode, *MaxSameSym* can also be used to govern how many positions can be opened in the same stock at the same time across multiple strategies.

When *MaxSameSym* > 1 for a single strategy, multiple positions in the same stock are treated as separate positions (as opposed to "adding to a position"). The exit-related formulas are applied to each sub-position separately. The trade list will show multiple entries and exits.

To model a system that scales in and/or out of positions, an alternative and generally better method is to use a separate strategy for each portion. See **Scaling In or Out of Positions** for details about that approach.

The **sample script martingale.rts** shows an interesting use of multiple positions in the same symbol (pyramiding).

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.223. MaxSetups

---

### Category

#### Strategy Elements

### Description

Caps the number of entry setups per day when a strategy has more setups than can be entered

### Input

Any formula that returns a count

### Notes

*MaxSetups* is applied before any other capacity constraints.

After building the list of symbols that meet the **EntrySetup** conditions, the list is sorted by **SetupScore**.

The setup list is then truncated at *MaxSetups*.

While it was often important to use *MaxSetups* in legacy mode, it has no significant use case in the default top-down mode.

(The insignificant use case would be to make tests run slightly faster by trimming the setup list before processing the setups.)

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.224. MDI

---

### Category

#### Indicator Functions

### Description

Wilder's Minus Directional Index

### Syntax

MDI(len)

### Parameters

len - lookback period

### Notes

This is the negative component of the **ADX** indicator, often referred to as -DI.

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.225. Median

---

### Category

#### Multi-Bar Functions

### Description

Median of a series of values

### Syntax

Median(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Values are calculated and sorted, then the middle one is returned.

## 17.12.226. Mid

---

### Category

#### String Functions

### Description

Return a segment of a string

### Syntax

Mid(string, start, length)

### Parameters

string - a **literal string** or **string function** result

start - the position of the first character to include, beginning at 0

length - the number of characters to include

## 17.12.227. Min

---

### Category

#### General-Purpose Functions

### Description

Smallest of a group of values

### Syntax

Min(value1, value2, ...)

### Parameters

value1 - any formula

value2 - any formula

... - any number of additional comma-separated formulas

### Notes

This function evaluates each of the formulas passed to it and returns the smallest value found.

## 17.12.228. MinN

---

## Category

### General-Purpose Functions

## Description

Nth Smallest of a group of values

## Syntax

MinN(N, value1, value2, ...)

## Parameters

N - number from 1 to count of values

value1 - any formula

value2 - any formula

... - any number of additional comma-separated formulas

## Notes

This function evaluates each of the formulas passed to it and returns the Nth smallest value found.

*MinN*(1, value1, value2) is the same as **Min**(value1, value2).

## 17.12.229. Month

---

## Category

### Bar Data Values

## Description

Current bar month number

Negative offsets, e.g. *Month*[-5], can be legitimately used to obtain the month of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best results when future dates are required, a **HolidayList** should also be provided.

## 17.12.230. NextOpen

---

## Category

### Bar Data Values

## Description

Next bar open price

## Notes

The ability to look ahead to the next open is included as a feature because it is often possible to learn approximately where a stock will open based on its pre-open live trading.

For example, if you enter long positions using limit orders and want to skip those entries where the gap down is larger than 5%, you could use *NextOpen* < 0.95 \* C as an **EntrySkip** formula.

## 17.12.231. NoNaN

---

## Category

## General-Purpose Functions

### Description

Evaluate an expression with no possibility a of NaN result

### Syntax

NoNaN(expr, replacement {0})

### Parameters

expr - any formula

replacement {0} - any formula

### Return Value

The result of the expression if it can be calculated, or *replacement* if it would result in NaN.

### Notes

The default (when only one argument) is to replace NaN with 0. You can optionally specify a different replacement value as needed.

It is not usually necessary to use *NoNaN* in any of the **strategy element** formulas. The backtest engine already equates "can't be evaluated" (NaN) with "false" or 0 in those formulas. The only exception is **EntrySkip**, for which "can't be evaluated" means "true" (*do skip the entry*).

## 17.12.232. Notes

---

### Category

**Script Sections**

### Description

A free-form section in which to organize any notes about the script

### Notes

This section of a script is not parsed. It is therefore not necessary to use **comments** to avoid syntax errors within this section.

## 17.12.233. NoWeekends

---

### Category

**Import Specification**

### Description

Whether to remove weekend bars from the data

### Choices

*False* - keep all bars (default)

*True* - remove weekend bars

### Notes

As of this writing, only crypto-currency data is known to have weekend bars.

The purpose of this option is to facilitate data alignment, e.g. for correlation studies between crypto and non-crypto series.

## 17.12.234. NumBars

---

### Category

#### Settings

### Description

Number of data bars to include in a scan or test

### Notes

Use in conjunction with either **StartDate** or **EndDate** to specify a bar count anchored to a date.

(If all three of these date settings are specified then *NumBars* will count back from *EndDate* and *StartDate* will be ignored.)

If a date range is not specified in a script then the dates from the **Settings Panel** will be used.

When **BarSize** is Daily, *NumBars* is a count of market days. Otherwise it is a count of weeks or months, depending on *BarSize*.

### Example

This is a convenient way to run a daily scan for live trading candidates:

```
▼ ScanSettings:  
  DataFile: daily_setup.rtd  
  EndDate: Latest  
  NumBars: 1  
  SaveAs: candidates.csv
```

## 17.12.235. OBV

---

### Category

#### Indicator Functions

### Description

On Balance Volume indicator

### Syntax

OBV(len)

### Parameters

len - lookback period

### Notes

Traditional OBV is simply the sum of up-bar volume minus the sum of down-bar volume for the specified period.

RealTest divides this value by the sum of all volume for the specified period to express it as a percentage.

This indicator could be written as  $(\text{Sum}((C > C[1]) * V, len) - \text{Sum}((C < C[1]) * V, len)) / \text{Sum}(V, len)$ .

## 17.12.236. Open or O

---

## Category

### Bar Data Values

## Description

Current bar open price

## Notes

Either *Open* or *O* can be used as the name of this value.

## 17.12.237. OpenSlip

---

## Category

### Strategy Elements

## Description

Slippage amount, in points (dollars per share or contract), for each transaction that simulates a market order filling at the open

## Input

Any formula specifying dollars per share or contract (points)

## Notes

Defines the amount of slippage to apply to each market-at-open transaction, in price points.

*OpenSlip* is applied to any transaction that logically occurs at the open and not at a specified limit or stop price.

If *OpenSlip* is not specified then **Slippage** is applied instead.

Note that *OpenSlip* will be applied to limit or stop orders when the fill is at the open but not at the limit or stop price, i.e., when the open gaps beyond the specified price.

## 17.12.238. OrderClerkFolder

---

## Category

### Settings or OrderSettings

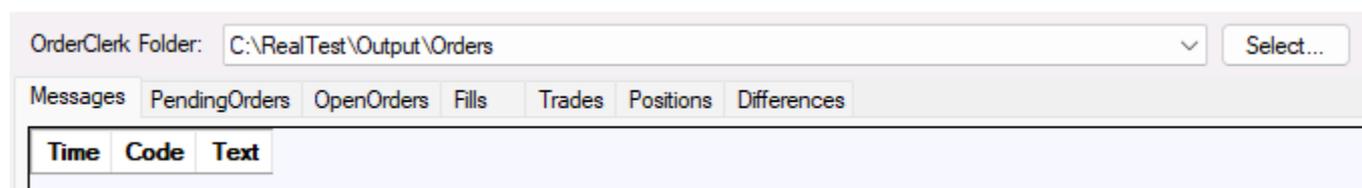
## Description

Path of the folder that **OrderClerk** uses for the strategies in this script.

## Notes

This setting is required to run a script as "Orders" when **OrdersMode** is *OrderClerk* or *OrderClerkStats*.

This will be the same folder that is selected at the top of the OrderClerk main window:



The screenshot shows a window titled "OrderClerk Folder:" with a text input field containing "C:\RealTest\Output\Orders" and a "Select..." button. Below this are several tabs: "Messages", "PendingOrders", "OpenOrders", "Fills", "Trades", "Positions", and "Differences". At the bottom, there is a table with three columns: "Time", "Code", and "Text".

RealTest will use the OrderClerkTrades.csv file in that folder as the TradeList to play back before generating tomorrow's orders, and will place the new order list in this folder to be processed by OrderClerk.

## 17.12.239. OrderMktAsLmtPct

---

### Category

#### Strategy Elements

### Description

Allows generated market orders to optionally be converted to limit orders X% beyond the last close.

### Value

A number expressing the desired percentage, e.g. 3 for 3%, or 0 if no conversion is desired

### Notes

The purpose of this strategy option is to allow what is logically a market-order to be executed as a marketable limit order.

This option provided to support order generation for exchanges that do not permit market orders, without having to convert the strategy to one that always uses limit orders.

Market orders are typically generated in the following situations:

- entry orders when **EntryTime** is *NextOpen* (or omitted) and neither **EntryLimit** nor **EntryStop** is specified
- exit orders when **ExitRule** is true and **ExitTime** is *NextOpen* (or omitted)
- exit orders when **ExitLimit** touches the last close and **ExitLimitTime** is *NextOpen*
- exit orders when **ExitStop** touches the last close and **ExitStopTime** is *NextOpen*

In all such cases, if *OrderMktAsLmtPct* returns a non-zero value then the order is converted to a LMT order.

For example, if 10 is the specified percentage, then a long entry (buy) order would use  $\text{Close} * 1.1$  as its limit price, and a long exit (sell) order would use  $\text{Close} * 0.9$  as its limit price. The idea is to allow plenty of extra room to ensure a fill while still using a limit order.

To selectively apply this option, e.g. in a strategy that trades on multiple exchanges, use a formula such as:

```
OrderMktAsLmtPct: if (?Exchange = "ASX", 10, 0)
```

This will tell RealTest to convert market orders to limit orders for stocks that trade on the ASX but not for stocks that trade elsewhere.

Note that this option only affects generated orders. Backtests still model the strategy with simple market order logic.

## 17.12.240. OrderNote / OrderExtra

---

### Category

#### Strategy Elements

### Description

String to add to the text of each order and as the "note" value in a **CSV Order Basket**

### Notes

Here is an example of how this works in a text order list:

Active Script - C:\RealTest\SCRIPTS\Examples\goal\_30\_15.rts

Template: **mr\_common**  
 Using: common  
 Quantity: mr\_alloc / mr\_pos  
 MaxSetups: mr\_pos - S.Positions  
 SetupScore: AtrPct  
 OrderNote: Format("(score={#})", AtrPct)

Log - C:\RealTest\Output\Orders\goal\_30\_15\_20230109\_orders.txt

```

*** ORDERS TO PLACE BEFORE THE OPEN OF 1/9/23 ***

mr_long exit orders -- change as needed to match actual positions
    (no exit orders)

mr_long entry orders
    (no setups)

mr_short exit orders -- change as needed to match actual positions
    cover 292 SHYF market on open (down day)

mr_short entry orders

short 62 (8%) [$7,989.32] NVCR 128.86 limit (score=11.59)
short 284 (8%) [$8,005.96] LOVE 28.19 limit (score=6.17)
short 85 (8%) [$8,014.65] WWE 94.29 limit (score=5.77)
short 77 (8%) [$7,963.34] MKSI 103.42 limit (score=4.90)
short 127 (8%) [$7,993.38] IRDM 62.94 limit (score=4.90)
short 304 (8%) [$7,995.20] XPOF 26.30 limit (score=4.88)
short 258 (8%) [$8,016.06] FRG 31.07 limit (score=4.80)
short 255 (8%) [$7,999.35] ICHR 31.37 limit (score=4.74)
short 26 (8%) [$7,955.74] RH 305.99 limit (score=4.64)
  
```

This shows how *OrderNote* might be used with **OrdersTemplate** to allow different strategies from the same script to send their orders to different linked IB accounts:

CSV File - C:\RealTest\Scripts\Examples\ib\_basket\_template.csv

#	Action	Quantity	Symbol	Exchange	Currency	TimelnForce	GoodTilDate	GoodAfterTime	OrderType	LmtPrice	AuxPrice	OcaGroup	OrderId	ParentOrderId	BasketTag	Account
1	act	qty	sym	exch	curr	tif	gtd	gat	type	lmt	stp	oca	id	parent	strat	note

Active Script - C:\RealTest\SCRIPTS\Examples\goal\_30\_15.rts\*

Template: **mr\_common**  
 Using: common  
 Quantity: mr\_alloc / mr\_pos  
 MaxSetups: mr\_pos - S.Positions  
 SetupScore: AtrPct  
 OrderNote: "DU123456"

Template: **moc\_common**  
 Using: moc\_alloc / moc\_pos  
 Quantity: moc\_pos  
 MaxSetups: moc\_pos  
 ExitRule: 1 // MOC  
 ExitTime: ThisClose  
 OrderNote: "DU654321"

Order List - C:\RealTest\Output\Orders\goal\_30\_15\_20230109.csv

#	Action	Quantity	Symbol	Exchange	Currency	TimelnForce	GoodTilDate	GoodAfterTime	OrderType	LmtPrice	AuxPrice	OcaGroup	OrderId	ParentOrderId	BasketTag	Account
1	BUY	292	SHYF	SMART/AMEX	USD	DAY			MKT			2			mr_short	DU123456
2	SELL	62	NVCR	SMART/AMEX	USD	DAY			LMT	128.86			3		mr_short	DU123456
3	SELL	284	LOVE	SMART/AMEX	USD	DAY			LMT	28.19			4		mr_short	DU123456
4	SELL	85	WWE	SMART/AMEX	USD	DAY			LMT	94.29			5		mr_short	DU123456
5	SELL	77	MKSI	SMART/AMEX	USD	DAY			LMT	103.42			6		mr_short	DU123456
6	SELL	127	IRDM	SMART/AMEX	USD	DAY			LMT	62.94			7		mr_short	DU123456
7	SELL	304	XPOF	SMART/AMEX	USD	DAY			LMT	26.30			8		mr_short	DU123456
8	SELL	258	FRG	SMART/AMEX	USD	DAY			LMT	31.07			9		mr_short	DU123456
9	SELL	255	ICHR	SMART/AMEX	USD	DAY			LMT	31.37			10		mr_short	DU123456
10	SELL	26	RH	SMART/AMEX	USD	DAY			LMT	305.99			11		mr_short	DU123456
11	BUY	340	ZI	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT	23.54			12	12	moc_long	DU654321
12	SELL	340	ZI	SMART/AMEX	USD	DAY			LMT	158.51		12	13	13	moc_long	DU654321
13	BUY	50	MDB	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT	88.07			14	14	moc_long	DU654321
14	SELL	50	MDB	SMART/AMEX	USD	DAY			LMT	220.11		14	15	15	moc_long	DU654321
15	BUY	91	BILL	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT	22.40			16	16	moc_long	DU654321
16	SELL	91	BILL	SMART/AMEX	USD	DAY			LMT	64.54		16	17	17	moc_long	DU654321
17	BUY	36	ENPH	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT						moc_long	DU654321
18	SELL	36	ENPH	SMART/AMEX	USD	DAY			LMT						moc_long	DU654321
19	BUY	358	PD	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT						moc_long	DU654321
20	SELL	358	PD	SMART/AMEX	USD	DAY			LMT						moc_long	DU654321
21	BUY	124	AXSM	SMART/AMEX	USD	GTD	20230109 15:40:00 US/Eastern		LMT						moc_long	DU654321

In addition to *OrderNote*, up to three other strategy-specific values can be added to an orders template.

The strategy formula names for these are *OrderExtra1*, *OrderExtra2*, and *OrderExtra3*.

These work the same as *OrderNote* except that their values will not appear in the text order list.

The CSV template second-row value names for these are *extra1*, *extra2*, and *extra3*.

## 17.12.241. OrderPrice

---

### Category

**Current Position Information**

### Description

Entry order price

### Notes

*OrderPrice* is the actual or implied price per share or contract of the entry order of this position.

For a strategy with **EntryLimit** and/or **EntryStop** this will be the limit or stop price that would have been used in an **order generated** by RealTest.

For a market-order strategy, *OrderPrice* is the **Close** of the most recently completed bar when the order is made.

Use **FillPrice** to obtain the actual entry price of the current position, which may differ from *OrderPrice* when there is a price gap between the close and next open.

## 17.12.242. OrderRank

---

### Category

**Current Position Information**

### Description

Returns the **TopDownMode** order rank number for this position's entry across all strategies.

### Notes

The top-down order ranking mechanism can be observed in detail by running a test with both *KeepTrades: Skipped* and *TestOutput: Log* enabled.

## 17.12.243. OrderSettings

---

### Category

**Script Sections**

### Description

Defines the settings to use only when the script **run mode** is **Orders**.

### Notes

The general-purose **Settings** section is always applied first, then modified by any items specified in *OrderSettings* when applicable.

See **Settings Sections** for details.

## 17.12.244. OrdersFile

---

### Category

**Settings** or **OrderSettings** or **Strategy Elements**

### Description

Path and name of the file to create when generating an order list as a text file, CSV basket file, or Alera signal file

### Notes

The settings-level *OrdersFile* can be used to override the default path and name for human-readable text and machine-readable CSV order list files.

The strategy-level *OrdersFile* is used to specify the location of an Alera Portfolio Manager signal file for each strategy.

See **File Path Specification** for helpful tips including special path expansion variables.

See **CSV Order Baskets** and/or **Alera Signal Files** for details about this setting.

## 17.12.245. OrdersInclude

---

### Category

**Script Sections**

### Description

Allows a script to include another script when run in **Orders Mode**

### Syntax

*OrdersInclude: path* where *path* is either a full file path (e.g. *C:\RealTest\Scripts\script.rts*) or a path relative to the Scripts folder (e.g. *Examples\script.rts*).

### Notes

See the general-purpose **Include** statement for further details.

## 17.12.246. OrdersLiveData

---

### Category

**Settings** or **OrderSettings**

### Description

Allows generation of intraday market orders when running in **Orders** mode

### Choices

*True* - generate today's live market orders

*False* - (default) generate tomorrow's orders as usual

### Notes

This is an advanced option provided for users who have found a way to import live "daily bars so far" just before the market close and wish to generate market orders for strategies that use **ThisClose** as their **EntryTime** and/or **ExitTime**.

## 17.12.247. OrdersMode

---

### Category

**Settings** or **OrderSettings**

### Description

Specifies which format to use when generating **Tomorrow's Orders**

### Choices

*Text* - (default) only generate human-readable text orders

*OrderClerk* - run the backtest in **Hybrid TradeList** mode using the **OrderClerkTrades.csv** in the specified **OrderClerkFolder** to play back past live trades, then generate an OrderClerk-compatible CSV order list in the same *OrderClerkFolder*.

*OrderClerkSync* - identical to *OrderClerk* mode unless there was a gap in time between the last trade in that mode and the end of available data, i.e., one or more recent days when orders were not generated. In that case, order playback switches to formula-based backtest for the missing period and additional orders are generated to re-synchronize the positions.

*OrderClerkStats* - run a backtest in **TradeList** mode using the **OrderClerkTrades.csv** file in the specified *OrderClerkFolder*, for the purpose of generating live trading stats to review. Note that this mode still must be run as "Orders" (not "Test") even though no orders are generated.

*Alera* - generate Alera Signal Files for each strategy that includes an **OrdersFile** specification.

*Rebalance* - generate orders for a rotational strategy in a format compatible with IB's **Portfolio Rebalance Tool**.

*Template* - use the CSV template specified in **OrdersTemplate** to generate an order list for use with IB Basket Trader or any other tool that requires a CSV-formatted order list which can be specified using this kind of template.

### Notes

See **Generated Order Types** for details about how RealTest maps strategy rules to generated orders.

To generate any kind of orders other than *Text*, *OrdersMode* must always be specified in **Settings** or **OrderSettings**.

Orders are only generated when a script is run by pressing the *Orders* button (not the *Test* button).

Text orders are also generated for every other mode unless *Always show text orders* is unchecked in **View / Program Options**

## 17.12.248. OrdersNetLiq

---

### Category

**Settings** or **OrderSettings**

### Description

Path and name of a text file that contains the current Net Liquidation Value (mark to market account balance) of your brokerage account

### Notes

This setting can be used to ensure that **orders are generated** with the correct **Quantity** (position size) for your current account value.

If you have a way to automatically recreate this one-line text file each day, or a habit of doing so manually, this is the simplest way to get your order sizes right.

The value found in the file becomes the value of **S.Alloc** for the purpose of order generation.

To clarify how this works, imagine that daily order generation for live trading began at the start of this year with an account balance of \$100K.

Your Settings might look like this:

```
▼ Settings:
  AccountSize: 100000
  StartDate: 1/1/22
  EndDate: Latest
```

When you generate tomorrow's orders, RealTest runs a backtest starting with \$100K on 1/1/22 (or the first market day thereafter), models all the trades that should have occurred, and accumulates their net P/L in each strategy's **S.Equity**.

On the last date of the test, orders are generated by evaluating all of the strategy rules to determine which open positions call for exits and which new setups call for entries in each strategy.

For each strategy that uses a typical *Quantity* formula with either explicit or implied *S.Alloc* as its basis, if *OrdersNetLiq* was provided, it will become the value of *S.Alloc*. Otherwise, *S.Alloc* will be whatever it is at the end of the backtest segment.

Examples of *Quantity* formulas that would use *OrdersNetLiq*:

```
Quantity: 10
QtyType: Percent
```

```
Quantity: S.Alloc / 10
QtyType: Value
```

```
Quantity: S.Alloc / 10 / FillPrice
QtyType: Shares
```

Examples of *Quantity* formulas that would NOT use *OrdersNetLiq*:

```
Quantity: 10000
QtyType: Value
```

```
Quantity: S.StartEquity / 10
QtyType: Value
```

```
Quantity: S.Equity / 10 / FillPrice
```

As always, position sizes for each backtest trade and generated order are calculated bottom-up, based on the *Quantity* formula provided.

The default calculation of *S.Alloc* is **Combined(S.Equity)**, which is always equivalent to the NLV of a model account running all of the strategies in the script using whatever their *Quantity* formulas happen to be.

If you use a different **Allocation** formula to override the default *S.Alloc*, or if you don't base your sizes on *S.Alloc* (whether explicitly or implicitly), then it would not make sense to use an *OrdersNetLiq* file.

Even if you are using the default *Allocation*, you may prefer to keep sizing new positions according to the system model rather than your live account balance. *OrdersNetLiq* simply gives you the choice of using live NLV, it's not a requirement.

## 17.12.249. OrdersTemplate

---

### Category

**Settings** or **OrderSettings**

### Description

Path and name of the CSV order list template file to use for generating this test's order basket

### Notes

See **CSV Order Baskets** for details about this setting.

The special keyword **rebalance** can be used in place of a file path to tell RealTest that you want to generate an import file for the **IB Rebalance Tool**.

## 17.12.250. OrderSum

---

### Category

**General-Purpose Functions**

### Description

Evaluates an expression for each non-skipped setup and returns the sum of the values

### Syntax

OrderSum(expression)

### Notes

This is a specialized function meant to be used in entry-related formulas that want to the sum of something across all current non-skipped setups (stocks for which an entry order will be placed).

*OrderSum(1)* can be used to count today's orders for the current strategy.

Use **SetupSum** to count or calculate something for all setups, not just the non-skipped ones.

Note that *OrderSum* cannot be used to refer to past setups in a strategy. It only has access to setups for the current date in the test. Therefore *OrderSum(expression)[offset]* will probably not have meaningful results.

## 17.12.251. PadAlignSym

---

### Category

**Import Specification**

### Description

Symbol to use for *Padding: AlignWithSymbol*

### Syntax

PadAlignSym: *symbol*

### Notes

The *AlignWithSymbol* **Padding** choice forces all imported stocks to have the same set of bar dates as the *PadAlignSym* stock by adding missing bars and removing extra bars.

This alignment is performed only within the existing date range of each stock. Extra bars are not added before the start or after the end of its data.

Since this setting is always simply one symbol, it should not be preceded by a \$ as is normally done to reference a symbol within a formula.

## 17.12.252. Padding

---

### Category

**Import Specification**

### Description

Import data padding type

## Syntax

Padding: *choice*

## Choices

*None* - don't add any padding bars (this is the default and can be used most of the time)

*AllMarketDays* - add a padding bar for any market day in a symbol's date range where the symbol has no bar

*AllWeekDays* - add a padding bar for any weekday in a symbol's date range where the symbol has no bar (including holidays)

*AllCalendarDays* - add a padding bar for any date in a symbol's date range where the symbol has no bar (including weekends)

*AlignWithSymbol* - add padding bars that the **PadAlignSym** symbol has but this symbol lacks, and remove bars from this symbol that the *PadAlignSym* lacks

## Notes

When Norgate is the data source, padding of any of the above types is provided by Norgate.

For other data sources, RealTest does the padding in the same way that Norgate does.

When data is imported for multiple markets that have different holiday schedules, "AllMarketDays" means any day where any of the included markets is open.

## 17.12.253. Parameters

---

### Category

**Script Sections**

### Description

Optimization Parameter Definitions

### Notes

See **Parameters Section** and **Optimization Dialog**.

## 17.12.254. PDI

---

### Category

**Indicator Functions**

### Description

Wilder's Plus Directional Index

### Syntax

PDI(len)

### Parameters

len - lookback period

### Notes

This is the positive component of the **ADX** indicator, often referred to as +DI.

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.255. Peak

### Category

#### Multi-Bar Functions

### Description

Value of the nth most recent peak of a series of prices or other values

### Syntax

Peak(expr, pctChg, nth {1})

### Parameters

expr - data series formula

pctChg - percent change required to delimit peaks and troughs

nth - which peak to locate (1, i.e., most recent if omitted)

### Notes

The definition of a Peak is the highest value which is then followed by one or more values that are at least n% below that most recent high.

Most technical analysis software (which is designed primarily for charting purposes) attaches Peak and Trough values to the bars on which they occur. RealTest (being designed for realistic backtesting) does not commit this look-ahead error.

Instead, the most recent peak is only returned starting with the subsequent bar that completes the n% drop following that peak. This makes it completely safe and legitimate to use these functions in backtesting.

The SPY chart below shows how this works. The bar marked by the pink vertical line is the first bar for which 417.39 is returned, even though that peak happened six bars earlier (as reflected in the **PeakBars** calculation).

The same rules apply to the **Trough** and **TroughBars** functions.

SPY - Daily 6/10/22: O = 394.88, H = 395.78, L = 389.75, C = 389.80 (-2.90%) Peak = 417.39 Trough = 389.46 PeakBars = 6 TroughBars = 15



## 17.12.256. PeakBars

## Category

### Multi-Bar Functions

## Description

Count of bars since the nth most recent peak of a series of prices or other values

## Syntax

PeakBars(expr, pctChg, nth {1})

## Parameters

expr - data series formula

pctChg - percent change required to delimit peaks and troughs

nth - which peak to locate (1, i.e., most recent if omitted)

## Notes

The definition of a Peak is the highest value which is then followed by one or more values that are at least n% below that most recent high.

See **Peak** for important additional information about how these functions work in RealTest.

## 17.12.257. PercentRank

---

## Category

### Multi-Bar Functions

## Description

Percent rank among values

## Syntax

PercentRank(expr, count)

## Parameters

expr - data series formula

count - lookback period

## Notes

*expr* is calculated for each bar, then results are sorted (ranked) from highest to lowest.

The highest value is given a **Rank** of 1, the next highest 2, and so on.

*PercentRank* is calculated as:  $100 / (\text{count}-1) * (\text{count}-\text{rank})$ , or simply 100 if count is 1.

For the highest value, this will always be 100.

For the lowest value, this will always be 0 (unless there is only one value).

For three values, *PercentRank* will always be 0, 50, or 100.

For four values, *PercentRank* will always be 0, 33.33, 66.66, or 100.

etc.

This implementation of PercentRank is identical to the function of the same name in Microsoft Excel.

For AmiBroker users, please note that their PercentRank function adds one to count, so if you're trying to exactly match their PercentRank(expr, count) for the same set of data, use PercentRank(expr, count + 1).

## 17.12.258. PercentRankN

---

### Category

#### Multi-Bar Functions

### Description

Value with a specific percent rank (the Nth percentile value)

### Syntax

```
PercentRankN(pctl, expr, count)
```

### Parameters

pctl - formula specifying a percent rank level

expr - data series formula

count - lookback period

### Notes

Performs the **PercentRank** function and then returns the value with the requested percentile.

## 17.12.259. PointValue

---

### Category

#### Stock/Contract Information

### Description

Futures contract point value

### Notes

Specifies the notional value of a 1-point change in price for a futures contract. This is also known as the "multiple".

Point values are obtained automatically when importing futures data from **Norgate**.

For CSV futures data import, it would be necessary to provide point values for each contract using a **SymInfo** file.

*PointValue* is assumed to always be \$1.00 for stocks.

## 17.12.260. PrevExitLimit

---

### Category

#### Current Position Information

### Description

Previous exit limit (target) price

### Notes

At position entry time, *PrevExitLimit* is set to 0.

On each bar that the position is open, after the **ExitLimit** formula is calculated, its value is placed in *PrevExitLimit*.

This makes it possible to implement a "trailing target", or any type of price target concept that needs to refer to its own prior value to be calculated.

(Previously this required using the **Data Section** and was therefore more complex.)

It is also possible for a strategy to refer to the current exit limit price of another strategy by using **Extern**(@other, PrevExitLimit).

## 17.12.261. PrevExitStop

---

### Category

#### Current Position Information

### Description

Previous exit stop price

### Notes

At position entry time, *PrevExitStop* is set to 0.

On each bar that the position is open, after the **ExitStop** formula is calculated, its value is placed in *PrevExitStop*.

This makes it possible to implement a "trailing stop", or any type of price stop concept that needs to refer to its own prior value to be calculated.

(Previously this required using the **Data Section** and was therefore more complex.)

It is also possible for a strategy to refer to the current exit stop price of another strategy by using **Extern**(@other, PrevExitStop).

### Examples

This is a very simple trend-following strategy with a 3\*ATR trailing stop:

```
Strategy: simple_trend
Side: Long
EntrySetup: c == hhv(c,100)
ExitStop: max(c - 3 * ATR(14), PrevExitStop)
```

## 17.12.262. Product

---

### Category

#### Multi-Bar Functions

### Description

Product of values

### Syntax

Product(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Evaluates *expr* for each of *count* bars and returns the product of the values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.263. PositionSum

---

### Category

#### General-Purpose Functions

### Description

Evaluates an expression for each currently open position and returns the sum of the values

### Syntax

PositionSum(expression)

### Notes

This is a specialized function meant to be used in entry-related formulas that need to know how many positions are open or will still be open after tomorrow's exits.

Referring to *PositionSum(1)* is equivalent to referring to *S.Positions* in that context.

Assuming a strategy uses *ExitRule: my\_rule* where *my\_rule* is an exit condition formula, and does exits at the next open (the default), *PositionSum(Not(my\_rule))* can be used to count the number of positions that will be open after tomorrow morning's exits but before any new entries. This kind of logic may be necessary when using a **TestScan** to produce a list of daily orders for the kind of strategy where you only want to use available position slots with no extra entry orders.

Other than the above example, in most use cases it is not necessary to deliberately calculate how many position slots will be available after tomorrow's exits. RealTest handles this automatically in both backtests and automatic order list generation (TestOutput: Orders). In these more typical scenarios, *S.Positions*, *PositionSum(1)*, and *PositionSum(Not(my\_rule))* will all return the same value.

Another possible use of PositionSum would be to calculate your own statistics about currently open positions, e.g. *PositionSum(Shares \* (C - FillPrice))* would (assuming a long position) tell you the current net mark-to-market value of your open positions (equivalent to S.M2M).

One more use of PositionSum might be to count how many other positions have the same characteristic (e.g. economic sector or industry) as the current one. For this purpose use the **This** function, e.g. *PositionSum(Top(InfoTRBC,2) = This(Top(InfoTRBC,2)))*.

Note that *PositionSum* cannot be used to refer to past positions in a strategy. It only has access to positions open on the current date in the test. Therefore *PositionSum(expression)[offset]* will probably not have meaningful results.

## 17.12.264. PriceRound

---

### Category

#### Strategy Elements

### Description

Specifies how to round trade prices in backtests and order lists

### Input

Any formula specifying a rounding interval (step value)

### Notes

If not specified, the default rounding interval is the **TickSize** of the symbol.

*PriceRound: 0.01* means round to the nearest penny (which is also the default *TickSize* if not known)

*PriceRound: 0* means don't do any rounding (include all available decimal places).

*PriceRound: if(C < 1, 0.0001, 0.01)* means round to nearest penny for stocks above \$1/share or nearest hundredth of a penny for penny stocks

Prices are rounded before being used to calculate trade profits/losses, which makes those figures more realistic.

Since RealTest always uses **split-unadjusted** (as-traded) prices in formulas, we do not need to worry about the extra decimal places that might be required for a past date of a stock that later had one or more splits. Prices by default are rounded to the as-traded *TickSize*.

Unless you have a specific need to control trade price rounding, it is recommended to NOT include this element in your Strategy definitions, so that the default (TickSize) rounding will be applied.

## 17.12.265. QtyFinal

---

### Category

#### Strategy Elements

### Description

Can be used to modify entry position sizes after top-down setup selection has been completed

### Notes

The initial desired **Quantity** of each setup must be specified prior to application of constraints such as **MaxExposure** or **MaxInvested**.

The final count of setups that can become orders is therefore not known yet when initial quantities are set.

*QtyFinal* is provided for cases where you want position sizes to be influenced by the number of orders that will be placed after constraints have been applied.

An example use case might be:

```
▽Parameters:
  maxpos:      10
  maxexp:     100

▽Strategy: my_strat
  Side:        Long
  MaxExposure: maxexp
  QtyType:     Percent

  // initial quantity (assumes all position slots will be used)
  Quantity:    maxexp / maxpos
  // evaluated along with EntrySetup, prior to final order selection

  // revised quantity (sizes up new positions to consume remaining exposure)
  QtyFinal:    (maxexp - 100 * S.Exposure) / S.EntryOrders
  // evaluated after final order selection -- count of new orders is in S.EntryOrders
```

This sets the initial position sizes to 10%.

After selecting today's setups while applying the *MaxExposure* constraint, positions are resized to use all remaining capital divided equally among the new orders to be placed (**S.EntryOrders**).

(Arbitrarily growing position sizes to use available exposure may increase position-specific risk and is probably not advisable, but if you wanted to test it, this would be a way to do so.)

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.266. QtyPrice

---

### Category

#### Strategy Elements

### Description

Specifies which price to use when calculating Quantity (Percent or Value types), trade fraction, and exposure

## Choices

OrderPrice - use the order price (default in top-down mode)

FillPrice - use the entry price (default in legacy mode)

## Notes

This new strategy-level setting replaces and augments the former *PercentOrder* and *ValueOrder* selections for **QtyType**.

The *QtyPrice* setting has several implications:

- which price to use when calculating **Quantity** for types other than *shares*
- how to interpret **FillPrice** on the day a position is entered, e.g. when used in **ExitLimit** or **ExitStop**
- which price to use when calculating **T.Fraction**
- which fraction value to therefore use when checking **MaxExposure** and calculating **S.Exposure** and **S.Usage**

In most cases the best practice will be to accept the default for this setting.

Using *QtyPrice: FillPrice* implies that you are able to size your positions and calculate your entry-day target and stop prices at the moment of entry based on the actual fill price or that you have automation software that can do so.

Using *QtyPrice: FillPrice* also implies that your rank-based final setup selection (depending on capacity constraints) is done at entry time.

## 17.12.267. QtyRound

---

### Category

**Strategy Elements**

### Description

Specifies how to round the result of the **Quantity** formula

### Input

Any formula specifying a rounding interval (step value)

### Notes

If not specified, the default rounding interval is 1 (whole shares).

*QtyRound: 0* means don't do any rounding (include all available decimal places).

*QtyRound: 10* means round to the next lower ten-share interval.

*QtyRound: 0.1* means round to the next lower tenth-of-a-share interval.

When rounded, Quantity is always rounded down (truncated), e.g. 12.99 will be rounded to 12 if *QtyRound: 1* is in use.

Quantity is rounded down by default to avoid the possibility of share rounding pushing a position value slightly over the MaxInvested threshold.

If you want to round up, or round nearest, specify *QtyRound: 0* and use the **Round** function in your *Quantity* formula while keeping **QtyType** as *Shares* (the default).

## 17.12.268. QtyType

---

### Category

**Strategy Elements**

## Description

Specifies how to interpret the **Quantity** formula

## Choices

Shares - *Quantity* represents the number of shares or contracts (default if not specified)

Value - *Quantity* represents the notional value of the position (e.g. dollars) based on entry fill price

Percent - *Quantity* represents a percentage of current allocation (**S.Alloc**)

## Notes

In the default top-down setup selection mode, **Quantity** is calculated and constraints (such as **MaxInvested**) are applied based on the **OrderPrice** of the **EntrySetup**.

In **Legacy Mode**, *Quantity* is based instead on **FillPrice**, implying that orders are placed when the market is open or about to open and the likely actual entry price can be known and used to calculate the position size.

To override either of the above defaults specify the **QtyPrice** choice explicitly.

The following are all equal position sizes in top-down mode:

```

▽Parameters:
  Positions: 10

▽Strategy: strat1
  Quantity: S.Alloc / Positions / OrderPrice
  QtyType: Shares

▽Strategy: strat2
  Quantity: S.Alloc / Positions
  QtyType: Value

▽Strategy: strat3
  Quantity: 100 / Positions
  QtyType: Percent
```

For *Legacy* mode substitute *FillPrice* for *OrderPrice* above.

## 17.12.269. Quantity

---

### Category

#### Strategy Elements

### Description

Specifies the number of shares or contracts to buy or sell short when opening a new position

### Input

Any formula specifying a number of shares or contracts, or a position value, or an allocation percentage

### Notes

The *Quantity* formula specifies the position size to use each trade in a strategy.

*Quantity* is calculated at setup selection time using the same current bar context as **EntrySetup**.

If there is no *Quantity* formula then a strategy will always invest 100% of current **Allocation** in each position.

If *Quantity* returns 0 then the entry is skipped and "zero quantity" is displayed as the skip reason.

*Quantity* can optionally be used to specify the **Side** (long vs. short) of a trade in a strategy that can go either way, such as a hedging strategy.

If **Side** is specified for a strategy, then the strategy always trades that side only and *Quantity* should always be expressed as a positive number.

If **Side** is not specified, then *Quantity* should return a positive number to enter a long position or a negative number to enter a short position.

See **QtyType** for important information about how *Quantity* is interpreted.

See **Asset Allocation and Position Sizing** for additional information.

## 17.12.270. Random

---

### Category

#### General-Purpose Functions

### Description

Returns a uniformly distributed random number

### Syntax

Random(min {0}, max {0}, step {0})

### Parameters

min {0} - the lowest value to possibly return

max {0} - the highest value to possibly return

step {0} - the smallest possible interval between return values

### Notes

If called with no parameters, i.e. *Random()*, the return value is a decimal number between 0 and 1.

To obtain a random integer between 1 and 100, use *Random(1,100,1)*.

To obtain a random limit price between 2% and 4% above yesterday's close, use *Random(c\*1.02, c\*1.04, 0.01)*.

## 17.12.271. RandomSeed

---

### Category

#### Settings

### Description

Permits use of the same sequence of pseudo-random numbers every time a script is run

### Notes

Pseudo-random numbers can be used explicitly by scripts via the **Random** function and are used implicitly in **Optimization** (some modes) and **Monte-Carlo analysis**.

If *RandomSeed* is not specified or is 0 then random numbers are different every time RealTest runs and are selected from a much larger set of possible values (the C runtime library function **rand\_s** is used).

When *RandomSeed* is specified, the provided value is passed to the C **srand** function and the **rand** function is then used for each random value needed.

It is recommended to **not** specify a *RandomSeed* unless you have a particular need for it, as the **rand\_s** function is significantly more robust than **rand** is.

## 17.12.272. Range or R

---

### Category

#### Bar Data Values

### Description

Current bar intraday range

### Notes

This is simply H-L of a bar.

Either *Range* or *R* can be used.

See also **TrueRange**.

## 17.12.273. Rank

---

### Category

#### Multi-Bar Functions

### Description

Numeric rank among values

### Syntax

Rank(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

*expr* is calculated for each bar, then results are sorted (ranked) from highest to lowest.

The highest value is given a rank of 1, the next highest 2, and so on.

See also **PercentRank**.

## 17.12.274. RankN

---

### Category

#### Multi-Bar Functions

### Description

Value with a specific numeric rank

### Syntax

RankN(rank, expr, count)

### Parameters

rank - formula specifying a rank number

expr - data series formula

count - lookback period

### Notes

Performs the **Rank** function and then returns the value with the requested rank.

## 17.12.275. Reduce

---

### Category

**Strategy Elements**

### Description

Specifies whether to reduce quantity when a new position's size would push total investment over the max investment cap

### Choices

*False* - don't reduce (default)

*True* - reduce

### Notes

By default, if the next position to be entered would put the total investment for that strategy above its **MaxExposure** and/or **MaxInvested** cap, the entry is skipped.

If *Reduce: True* is specified, then rather than skipping the entry, its **Quantity** is reduced to allow it to fit within the cap.

## 17.12.276. Replace

---

### Category

**String Functions**

### Description

Replace all instances of a string within another string

### Syntax

Replace(string, find, replace)

### Parameters

string - a **literal string** or **string function** result

find - the string (or function result) to find

replace - the string (or function result) to replace it with

## 17.12.277. Results

---

### Category

**Script Sections**

### Description

Results column definitions

### Notes

See **Results Section** and **Results Windows**.

## 17.12.278. ResultsFile

---

### Category

#### Settings

### Description

Path and name of a results file (RTR) to open or create before a test is run, and save at the end of the run

### Notes

By default when a test is run, RealTest uses the most recently opened **results window**, or creates a new untitled one for the purpose.

This setting allows you to choose a specific RTR file to use when running tests with this script.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.279. Right

---

### Category

#### String Functions

### Description

Return the right end of a string

### Syntax

Right(string, length)

### Parameters

string - a **literal string** or **string function** result

length - the number of characters to include

## 17.12.280. RiskFreeRateSym

---

### Category

#### Settings

### Description

Symbol of an imported data series to store in the test statistics for later use when calculating Sharpe or other stats that require this.

### Notes

The daily closing values of this data series are stored in **S.RiskFreeRate** when a test is run.

This in turn is used in the calculation of Sharpe Ratio in the default Results.rts formula:

Sharpe:  $\{ \#2 \} \text{SQR}(S.BPY) * \text{Avg}((S.NetPct - S.RiskFreeRate / S.BPY), \text{Periods}) / \text{StdDev}(S.NetPct, \text{Periods})$

For this to work correctly the data must be a rate series, e.g. close=5 means 5%, as in the Norgate symbols **%FFYE** or **%3MTCM**.

Since this setting is always simply one symbol, it should not be preceded by a \$ as is normally done to reference a symbol within a formula.

The referenced symbol must also have been added to the **Import** of the current data file.

## 17.12.281. ROC or PctChg

---

### Category

#### Multi-Bar Functions

### Description

%gain/loss

### Syntax

ROC(expr, count) or PctChg(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *ROC* or *PctChg* can be used as the name of this function.

Return value is equivalent to  $100 * (expr / expr[count] - 1)$ .

## 17.12.282. Round

---

### Category

#### General-Purpose Functions

### Description

Round value to nearest step

### Syntax

Round(value, step {1}, direction {0})

### Parameters

value - formula specifying a numeric value

step - formula specifying a rounding interval (optional, defaults to 1)

direction - formula specifying a rounding direction (optional, defaults to 0)

### Notes

The *direction* parameter is interpreted as follows:

- 0 means round to the nearest *step*, e.g. if step is 1, round UP if the decimal part of value is  $\geq 0.5$  else round down
- 1 means always round up
- -1 means always round down

For example, to round a calculated price to the nearest penny, use *Round(price, 0.01)*.

Note that negative values are always rounded "away from zero", e.g.

- $\text{round}(-1.5, 1, 0) = -2$
- $\text{round}(-1.5, 1, 1) = -2$
- $\text{round}(-1.5, 1, -1) = -1$

In other words negative values are rounded as if they were positive and then the minus sign is applied to the result.

(This is consistent with how `=Round`, `=RoundUp`, and `=RoundDown` work in Excel.)

## 17.12.283. RSI

---

### Category

#### Indicator Functions

### Description

Wilder's relative strength index

### Syntax

`RSI(len)`

### Parameters

len - lookback period

### Notes

Calculation uses the original Welles Wilder formula. Wilder's exponential smoothing is equivalent to using  $2*len-1$  in a regular exponential moving average.

It's a little-known fact  $RSI(len)$  crossing above or below 50 is the same as  $C$  crossing above or below  $EMA(C, 2*len-1)$ .

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.284. RRSI

---

### Category

#### Indicator Functions

### Description

Reverse RSI (price required for RSI to reach level)

### Syntax

`RRSI(len, level)`

### Parameters

len - lookback period

level - formula specifying the desired RSI level

### Notes

This function can be used to calculate entry or exit limit prices based on RSI reaching a specific level.

Calculation uses the original Welles Wilder formula.

Wilder's exponential smoothing is equivalent to using  $2^{*len-1}$  in a regular exponential moving average.

This indicator supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable length.

## 17.12.285. RsiF

---

### Category

#### Multi-Bar Functions

### Description

RSI as a function

### Syntax

RsiF(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

The standard **RSI** indicator always uses the series of closing prices for its calculations.

This function makes it possible to calculate RSI for any series of values.

Calculation uses the original Welles Wilder formula.

Wilder's exponential smoothing is equivalent to using  $2^{*len-1}$  in a regular exponential moving average.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.286. S.Alloc

---

### Category

#### Daily Test Statistics

### Description

The amount of money allocated to this strategy on this test date

### Notes

*S.Alloc* is the most recently calculated value of the **Allocation** formula.

If no *Allocation* formula was given, *S.Alloc* defaults to **Combined(S.Equity)**.

*S.Alloc* is calculated once per backtest date, before any trade entries or exits are processed. Think of it as the "overnight" account value.

Use **S.Alloc** to access your current allocation value in the **Quantity** (position size) formula, if your **QtyType** is *Shares* (default) or *Value*.

If *QtyType* is *Percent*, then *S.Alloc* is automatically used as the value of which to calculate the

percentage.

In contrast to *S.Alloc*, *S.Equity* is updated every time a trade exit is processed. Since exits are typically processed before entries, *S.Alloc* may therefore be slightly different from *Combined(S.Equity)* if referenced in strategy entry-related formulas such as *Quantity*. Using *S.Alloc* rather than *Combined(S.Equity)* in these formulas ensures that backtest position sizes match what the order quantities would have been without knowledge of that morning's exit details.

*S.Alloc* is used in many of the default test statistics formulas provided in the **Results** and **Graphs** scripts.

It is also used internally as the denominator of built-in stats such as **S.DDPct**, **S.MaxDDPct**, and **S.NetPct**.

See **Asset Allocation and Position Sizing** for additional information.

## 17.12.287. S.BPx

---

### Category

#### Daily Test Statistics

### Description

Number of stat bars (dates) per period in the current test stats data

### Notes

Specific values are:

- S.BPY = bars per year
- S.BPQ = bars per quarter
- S.BPM = bars per month
- S.BPW = bars per week

The size of each stat period in a test will always be the smallest bar size used in any strategy.

For example, if a script includes a daily bar strategy and a weekly bar strategy, there will be 252 stat periods per year for both strategies, and *S.BPY* will return 252.

If a script only includes weekly bar strategies, there will be 52 stat periods per year and *S.BPY* will return 52.

The purpose of *S.BPY* is to facilitate calculation of formulas such as ROR that need to "annualize" a value (see *Results.rts*).

The purpose of the other intervals is to facilitate rolling return graphs of various durations (see *Graphs.rts*).

Because these are test-level statistic, the values returned will be the same for every strategy regardless of strategy-specific bar size.

## 17.12.288. S.CashInOut

---

### Category

#### Daily Test Statistics

### Description

The running total net **CashInOut** for this strategy as of this date

### Notes

The net cash in-out for one stats period can be calculated as *S.CashInOut* - *S.CashInOut[1]*

## 17.12.289. S.Comms

---

### Category

#### Daily Test Statistics

### Description

The sum of all **commissions** paid by this strategy on this test date

## 17.12.290. S.Compounded

---

### Category

#### Daily Test Statistics

### Description

Stats **compounding** flag

### Notes

Returns 1 (true) if the test used compounding, or 0 (false) otherwise.

A test uses compounding if the **Allocation** or **Quantity** formula of any strategy refers to **S.Equity**.

If neither formula refers to S.Equity in any strategy then the test does not use compounding.

If either formula is absent in any strategy, then the test *does* use compounding, because the default formula for both of these strategy elements is, in fact, *S.Equity*.

Because this is a test-level statistic, the value returned will be the same for every strategy (hence no need for "Combined").

## 17.12.291. S.Date

---

### Category

#### Daily Test Statistics

### Description

The current test date

## 17.12.292. S.DDBars

---

### Category

#### Daily Test Statistics

### Description

The duration, in bars, of the current drawdown for this strategy as of this test date

## 17.12.293. S.DDDlr

---

### Category

## Daily Test Statistics

### Description

The size, in dollars, of the current drawdown for this strategy as of this test date

### Notes

The internal daily equity value used to calculate drawdown includes the mark-to-market value of open positions.

## 17.12.294. S.DDPct

---

### Category

#### Daily Test Statistics

### Description

The size, in percentage, of the current drawdown for this strategy as of this test date

### Notes

The internal daily equity value used to calculate drawdown includes the mark-to-market value of open positions.

See **Compounding** for details on how drawdown percentage is calculated for compounded vs. non-compounded tests.

## 17.12.295. S.Dividends

---

### Category

#### Daily Test Statistics

### Description

The sum of all **dividends** received or paid by this strategy on this test date

### Notes

Total dividends in a backtest can be added to Results.rts as *Dividends: Sum(S.Dividends, S.Number)*.

## 17.12.296. S.Entries

---

### Category

#### Daily Test Statistics

### Description

The number of new positions entered by this strategy on this test date

### Notes

If referenced from any entry-related strategy formula, *S.Entries* will return the prior test day's entry count, not that of the current day.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.297. S.EntryOrders

---

### Category

#### Daily Test Statistics

### Description

The number of entry orders placed on this test date

### Notes

*S.EntryOrders* becomes available at the end of the daily setup selection and ranking process.

It can be used in the **QtyFinal** formula as needed to change position sizes based on order count vs. exposure capacity.

*S.EntryOrders* can also be used as a statistic showing how many orders were "placed" each day.

For strategies that enter "at market" *S.EntryOrders* will usually be the same as **S.Entries**.

For strategies that enter using stop and/or limit orders, *S.EntryOrders* will usually be more than *S.Entries*.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.298. S.Equity

---

### Category

#### Daily Test Statistics

### Description

The current account value of a strategy or group of strategies as of this test date

### Notes

By default, *S.Equity* includes the mark-to-market value, **S.M2M**, of currently open positions. To get closed-trade-only equity, use *S.Equity - S.M2M*.

If **MarkToMarket** is set to *False*, then *S.Equity* does not include *S.M2M*, and will only change value when a position is closed.

*S.Equity* also includes net cash deposits and withdrawals (from the **CashInOut** strategy element). To factor those out, use *S.Equity - S.CashInOut*.

Use **Combined** to access the combined equity value of all strategies, i.e., *Combined(S.Equity)*, which is also the default value of **S.Alloc** if a different **Allocation** was not specified.

Use **Extern** to access the combined equity of a specific group of strategies defined by a **StatsGroup**, e.g, *Extern(@group\_name, S.Equity)*.

## 17.12.299. S.Exits

---

### Category

#### Daily Test Statistics

### Description

The number of positions exited by this strategy on this test date

## 17.12.300. S.Exposure

---

### Category

#### Daily Test Statistics

### Description

Total fraction of allocation held overnight in open positions by this strategy on this test date

### Notes

*S.Exposure* is calculated as the sum of **FillFraction** for each position held overnight each day.

*S.Exposure*, like *FillFraction*, is always positive, regardless of side.

To calculate net long-short exposure while a test is running, use **PositionSum**(*Side* \* *FillFraction*).

When referenced from entry-related formulas in a strategy that enters positions at the open or at the close, *S.Exposure* will have been reduced by any exits that occurred at that same time.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.301. S.First

---

### Category

#### Daily Test Statistics

### Description

The number of the first stat period in a test in which a position was entered

### Notes

In the default **Results Section**, "Periods" is defined as  $S.Number - S.First + 1$ , and is then used in all of the stats calculations that require a test period count.

By using *Periods* in this way, the stats accurately reflect the span of time which actual trading occurs, and omit any "warm-up" bars where indicators are still being calculated.

The **Combined** *S.First* value is the smallest value for any strategy (benchmark strategies are not included in combined stats).

In the **Summary Report**, the individual strategy stats will each reflect that strategy's number of periods (assuming default results.rts formulas).

The bar size units of *S.First* will always be the test-level bar size, not the strategy-specific bar size.

## 17.12.302. S.Interest

---

### Category

#### Daily Test Statistics

### Description

The net interest received and/or paid for excess cash and/or margin loan on this date

### Notes

*S.Interest* is only relevant in the **Combined** stats series. It will always be 0 for individual strategies.

Total net interest in a backtest can be added to Results.rts as *Interest: Sum(S.Interest, S.Number)*.

See **CashIntPct**, **MarginIntPct**, and **RiskFreeRateSym** for details on how interest received

and/or charged is calculated and applied.

## 17.12.303. S.Invested

---

### Category

**Daily Test Statistics**

### Description

Net dollars held overnight in open positions by this strategy on this test date

### Notes

*S.Invested* is calculated as the sum of (side \* shares \* entry price) for each open position.

If a strategy is long-only, the value is always positive.

If a strategy is short-only, the value is always negative.

If a strategy trades both sides, the value reflects the long/short balance (will be near zero if both sides are fully invested).

When referenced from entry-related formulas in a strategy that enters positions at the open or at the close, *S.Invested* will have been reduced by any exits that occurred at that same time.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.304. S.LossBars

---

### Category

**Daily Test Statistics**

### Description

The sum of durations, in bars, for trades exited as losses by this strategy on this test date

## 17.12.305. S.LossDlr

---

### Category

**Daily Test Statistics**

### Description

The sum of dollar losses for all trades exited as losses by this strategy on this test date (expressed as a positive value)

## 17.12.306. S.Losses

---

### Category

**Daily Test Statistics**

### Description

The number of trades exited as losses by this strategy on this test date

## 17.12.307. S.LossPct

---

### Category

**Daily Test Statistics**

### Description

The sum of *dollar loss / dollar position size* for all trades exited as losses in this strategy on this test date (expressed as a positive value)

### Notes

Use **S.LossPctAlloc** if you need the sum of *dollar loss / strategy allocation*.

## 17.12.308. S.LossPctAlloc

---

### Category

**Daily Test Statistics**

### Description

The sum of *dollar loss / strategy allocation* for all trades exited as losses in this strategy on this test date (expressed as a positive value)

### Notes

Use **S.LossPct** if you need the sum *dollar loss / dollar position size*.

## 17.12.309. S.M2M

---

### Category

**Daily Test Statistics**

### Description

Net gain or loss, in dollars, of all positions remaining open in this strategy on this test date

### Notes

The default "M2M" **daily stats graph** is calculated as  $S.M2M / S.Alloc$  and displayed as a percentage.

## 17.12.310. S.MAE

---

### Category

**Daily Test Statistics**

### Description

Worst-case ("adverse") net gain or loss, in dollars, of all positions remaining open in this strategy on this test date

### Notes

This is theoretically the worst daily net P&L (M2M) value that could have occurred during this day, had all positions touched their least favorable prices simultaneously.

The default "M2MA" **daily stats graph** is calculated as  $S.M2MA / S.Alloc$  and displayed as a percentage.

## 17.12.311. S.MFE

---

### Category

**Daily Test Statistics**

### Description

Best-case ("favorable") net gain or loss, in dollars, of all positions remaining open in this strategy on this test date

### Notes

This is theoretically the best daily net P&L (M2M) value that could have occurred during this day, had all positions touched their most favorable prices simultaneously.

The default "M2MF" **daily stats graph** is calculated as  $S.M2MF / S.Alloc$  and displayed as a percentage.

## 17.12.312. S.MaxAlloc

---

### Category

**Daily Test Statistics**

### Description

The amount, in dollars, of the largest allocation value of this strategy as of this test date.

## 17.12.313. S.MaxDDBars

---

### Category

**Daily Test Statistics**

### Description

The duration, in bars, of the longest drawdown as of this test date

### Notes

The longest drawdown is not necessarily the largest one.

## 17.12.314. S.MaxDDDlr

---

### Category

**Daily Test Statistics**

### Description

The size, in dollars, of the largest drawdown of this strategy as of this test date

### Notes

The internal daily equity value used to calculate drawdown includes the mark-to-market value of

open positions.

## 17.12.315. S.MaxDDPct

---

### Category

#### Daily Test Statistics

### Description

The size, in percentage, of the largest drawdown of this strategy as of this test date

### Notes

The internal daily equity value used to calculate drawdown includes the mark-to-market value of open positions.

See **Compounding** for details on how drawdown percentage is calculated for compounded vs. non-compounded tests.

## 17.12.316. S.MaxEquity

---

### Category

#### Daily Test Statistics

### Description

The amount, in dollars, of the largest net liquidation value of this strategy as of this test date.

### Notes

See **S.Equity** for notes on what is included in this value.

## 17.12.317. S.MinAlloc

---

### Category

#### Daily Test Statistics

### Description

The amount, in dollars, of the smallest allocation value of this strategy as of this test date.

## 17.12.318. S.MinEquity

---

### Category

#### Daily Test Statistics

### Description

The amount, in dollars, of the smallest net liquidation value of this strategy as of this test date.

### Notes

See **S.Equity** for notes on what is included in this value.

## 17.12.319. S.NetDlr

---

## Category

### Daily Test Statistics

## Description

The daily dollar change in net liquidation value of the portion of the account **allocated** to this strategy as of this test date

## Notes

This is equivalent to  $S.Equity - S.Equity[1]$ .

## 17.12.320. S.NetFx

---

## Category

### Daily Test Statistics

## Description

The sum of net currency exchange rate change impact on trade profit or loss (**T.NetFx**) for all trades closed this period

## Notes

The value will always be 0 unless your test was correctly set up for multi-currency strategy modeling and one or more stocks traded had different base currency than your account's base currency.

See **Currency** and **Testing Multi-Currency Strategies** for details on how this works.

## 17.12.321. S.NetPct

---

## Category

### Daily Test Statistics

## Description

The daily percent change in net liquidation value of the portion of the account **allocated** to this strategy as of this test date

## Notes

This is equivalent to  $(S.Equity - S.Equity[1]) / S.Alloc[1]$ .

## 17.12.322. S.Number

---

## Category

### Daily Test Statistics

## Description

The number of this stat period in a test

## Notes

In the context of the **Graphs Section**, *S.Number* is also the count of stat periods so far as of this test date.

In the context of the **Results Section**, *S.Number* is also the total number of stat periods in the test.

The size of each stat period in a test will always be the smallest bar size used in any strategy.

For example, if a script includes a daily bar strategy and a weekly bar strategy, and is run for one year, there will be 252 stat periods for both strategies, and *S.Number* will be 252 for the final period of either strategy.

If a script only includes weekly bar strategies and is run for one year, there will be 52 stat periods and *S.Number* will be 52 for the final period.

Because this is a test-level statistic, the value returned will be the same for every strategy (hence no need for "Combined").

## 17.12.323. S.Positions

---

### Category

#### Daily Test Statistics

### Description

The number of positions that are currently open in the current strategy.

### Notes

When referenced from entry-related formulas in a strategy that enters positions at the open or at the close, *S.Positions* will have been reduced by any exits that occurred at that same time, allowing it to effectively be used in your entry-related strategy formulas to may need to calculate the number of available "position slots".

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.324. S.RiskFreeRate

---

### Category

#### Daily Test Statistics

### Description

The daily closing value of the data series specified by the **RiskFreeRateSym** symbol

### Notes

This is mainly useful when calculating the Sharpe Ratio, as in the default formula from Results.rts:

Sharpe:  $\{ \#2 \} \text{SQR}(S.BPY) * \text{Avg}((S.NetPct - S.RiskFreeRate / S.BPY), \text{Periods}) / \text{StdDev}(S.NetPct, \text{Periods})$

If *RiskFreeRateSym* is not specified or if it refers to a symbol that is not available in the current data file, *S.RiskFreeRate* will be a series of zeros.

## 17.12.325. S.Setups

---

### Category

#### Daily Test Statistics

### Description

The number of stocks that met the **EntrySetup** criteria in the current strategy.

### Notes

If referenced from any entry-related strategy formula, *S.Setups* will return the prior test day's setup count, not that of the current day.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

## 17.12.326. S.Slips

---

### Category

**Daily Test Statistics**

### Description

The sum of all **slippage amounts** applied to trade entries or exits by this strategy on this test date

## 17.12.327. S.StartEquity

---

### Category

**Daily Test Statistics**

### Description

The starting equity, in dollars, for this strategy

### Notes

This value will be the same for every test date.

See **Allocation** for more information.

## 17.12.328. S.Stops

---

### Category

**Daily Test Statistics**

### Description

The number of trades that exited because of their **ExitStop** for this strategy on this test date

## 17.12.329. S.Targets

---

### Category

**Daily Test Statistics**

### Description

The number of trades that exited because of their **ExitLimit** for this strategy on this test date

## 17.12.330. S.TradeBars

---

## Category

### Daily Test Statistics

## Description

The sum of durations, in bars, for all trades exited by this strategy on this test date

## 17.12.331. S.TradeDlr

---

## Category

### Daily Test Statistics

## Description

The sum of dollar gains-losses for all trades exited in this strategy on this test date

## 17.12.332. S.TradePct

---

## Category

### Daily Test Statistics

## Description

The sum of *net dollar gain or loss / trade position size* for all trades exited in this strategy on this test date

## Notes

Use **S.TradePctAlloc** if you need the sum of *dollar gain or loss / strategy allocation*.

## 17.12.333. S.TradePctAlloc

---

## Category

### Daily Test Statistics

## Description

The sum of *net dollar gain or loss / strategy allocation* for all trades exited in this strategy on this test date

## Notes

Use **S.TradePct** if you need the sum of *dollar gain or loss / trade position size*.

## 17.12.334. S.TWEQ

---

## Category

### Daily Test Statistics

## Description

The time-weighted equity of this strategy as of this test date

## Notes

If a test uses compounded stats (if **S.Compounded** is 1) then *S.TWEQ* is a compounded series of daily returns.

In this case, *S.TWEQ* starts a \$1 and then is multiplied each day by  $(1 + \text{that day's percent return})$ ,

resulting in a "growth of \$1" series.

When a test does not use compounding, S.TWEQ is the same as **S.Equity**.

## 17.12.335. S.Usage

---

### Category

**Daily Test Statistics**

### Description

Peak fraction of allocation used in open positions by this strategy on this test date

### Notes

*S.Usage* is calculated as the sum of **FillFraction** for each position that was open at any time this day.

*S.Usage*, like *FillFraction*, is always positive, regardless of side.

The purpose of this stat is to show the maximum intraday capital usage required to trade the strategy (or **Combined** strategies).

To constrain a strategy or set of strategies to not exceed a *S.Usage* cap:

- in the default top-down mode use **MaxExposure**
- in **Legacy Mode** use an **EntrySkip** formula that references *S.Usage*

## 17.12.336. S.WinBars

---

### Category

**Daily Test Statistics**

### Description

The sum of durations, in bars, for all trades exited as wins in this strategy on this test date

## 17.12.337. S.WinDlr

---

### Category

**Daily Test Statistics**

### Description

The sum of dollar gains for all trades exited as wins in this strategy on this test date

## 17.12.338. S.WinPct

---

### Category

**Daily Test Statistics**

### Description

The sum of *dollar gain / trade position size* for all trades exited as wins in this strategy on this test date

## Notes

Use **S.WinPctAlloc** if you need the sum of *dollar gain / strategy allocation*.

## 17.12.339. S.WinPctAlloc

---

### Category

**Daily Test Statistics**

### Description

The sum of *dollar gain / strategy allocation* for all trades exited as wins in this strategy on this test date

### Notes

Use **S.WinPct** if you need the sum of *dollar gain / trade position size*.

## 17.12.340. S.Wins

---

### Category

**Daily Test Statistics**

### Description

The number of trades exited as wins by this strategy on this test date

## 17.12.341. SAR

---

### Category

**Indicator Functions**

### Description

Wilder's Parabolic Stop And Reverse

### Syntax

SAR(accel {0.02}, max {0.2}, len {100})

### Parameters

accel {0.02} - initial acceleration factor

max {0.2} - maximum acceleration factor

len {100} - number of bars to use in the calculation

### Notes

Implements the **Parabolic Time/Price System** as an indicator function for use in trading strategies.

The standard SAR is calculated using price highs and lows.

To calculate SAR using a single non-standard value series, use **SarF**.

## 17.12.342. SarF

---

## Category

### Multi-Bar Functions

## Description

SAR as a function

## Syntax

```
SarF(expr, accel {0.02}, max {0.2}, count {100})
```

## Parameters

expr - data series formula

accel {0.02} - initial acceleration factor

max {0.2} - maximum acceleration factor

count {100} - number of bars to use in the calculation

## Notes

Use *SarF* to calculate SAR using a single non-standard value series.

Use **SAR** for the standard indicator based on price highs and lows.

## 17.12.343. SaveAs

---

## Category

### Import Specification

## Description

Path and name of the data file (.RTD) to save at the end of an import

## Notes

If the *SaveAs* file already exists, it will be overwritten without confirmation.

If *SaveAs* is not specified, RealTest will show a standard file save dialog at the end of the import.

## 17.12.344. SaveChartsTo

---

## Category

### Settings

## Description

Path of a folder in which to automatically save a chart for every row of the scan

## Notes

If this setting is present, then every time a scan is run, a chart will be automatically created for every item (row) of the scan.

Charts are created in the specified folder using *symbol\_date.png* (e.g. MSFT\_20210518.png) as their file names.

Existing folder contents are not deleted before new charts are added, but existing chart files with the same names will be overwritten without asking.

Saved charts have the same width and height and other display options as the most recently viewed chart window.

This feature was added to aid discretionary traders in creating a "chart book" for study purposes.

Charts for all items in a scan that has already been run can be saved via the **Scan Menu**.

## 17.12.345. SavePositionsAs

---

### Category

**Settings**

### Description

Path and name of a CSV file to optionally produce at the end of a test to list positions that were still open

### Notes

If *SavePositionsAs* is specified, on the last date of a backtest, before the "end of test" exits are processed, a list of open positions in all strategies is created and saved to the specified CSV file.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.346. SaveScanAs

---

### Category

**Settings**

### Description

Path and name of CSV file to optionally create

### Notes

If *SaveScanAs* is specified, then the contents of the scan window are automatically saved to the specified file in CSV format each time a scan is run. If the file already exists, it will be overwritten without confirmation.

If *SaveScanAs* is not specified, there is no prompt to save a scan, but you can easily do so using the **Scan Menu**.

Most scans that you run are likely to be quick one-off explorations that do not require saving. The *SaveScanAs* setting was added mainly to make it easier to create a daily candidate list for live trading.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.347. SaveStatsAs

---

### Category

**Settings**

### Description

Path and name of CSV file to optionally create

### Notes

If *SaveStatsAs* is specified, then all formulas in the **Graphs** section are evaluated for each date of the test and written to a CSV file. If the file already exists, it will be overwritten without confirmation.

The CSV file will contain a separate row per date for each individual **Strategy**, each **StatsGroup**, and **Combined**.

The columns in each row are Date and Strategy followed by a column for each *Graphs* item.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.348. SaveTestListAs

---

### Category

**Settings**

### Description

Path and name of CSV file to optionally create

### Notes

If *SaveTestListAs* is specified, then at the end of a script run the visible contents of the Results Window are saved as a CSV file. If the file already exists, it will be overwritten without confirmation.

This is equivalent to manually selecting *Save List as CSV File* from the **Results Menu** after the test or optimization run finishes.

The purpose of this setting is to facilitate automation of optimization runs with summary stats saved in CSV format for external processing.

To automatically save the detailed stats from a single test to CSV, use **SaveStatsAs**.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.349. SaveTradesAs

---

### Category

**Settings**

### Description

Path and name of CSV file to optionally create

### Notes

If *SaveTradesAs* is specified, then all standard and custom trade items for every trade in the test are written in CSV columns as raw values (format codes are ignored). If the file already exists, it will be overwritten without confirmation.

The content of this CSV file is the same as would be produced by opening the **Trade List** for the test and then saving it in CSV format.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

See **File Path Specification** for helpful tips including special path expansion variables.

## 17.12.350. SaveTradesType

---

### Category

## Settings

### Description

Format to use when creating the **SaveTradesAs** output file

### Notes

Choices are *Full* or *Compact*.

If *Full* is specified or if **SaveTradesType** is not specified, then all of the **Trade List** columns are written to the CSV file.

If *Compact* is specified then the CSV file is written with only the fields required for **Imported Trade List** playback.

Unlike some **Scan and Test Settings**, there is no **Settings Panel** equivalent for this particular option and the setting does not persist.

## 17.12.351. Scan

---

### Category

#### Script Sections

### Description

Scan definition

### Notes

See **Scan Section** for details.

## 17.12.352. ScanInclude

---

### Category

#### Script Sections

### Description

Allows a script to include another script when run in **Scan Mode**

### Syntax

*ScanInclude: path* where *path* is either a full file path (e.g. *C:\RealTest\Scripts\script.rts*) or a path relative to the Scripts folder (e.g. *Examples\script.rts*).

### Notes

See the general-purpose **Include** statement for further details.

## 17.12.353. ScanNoDefCols

---

### Category

#### Settings

### Description

Allows the default Date and Symbol columns to optionally be omitted from **Scan Window** and **SaveScanAs** output

### Notes

By default every scan begins with Date and Symbol columns, followed by the columns defined in the Scan section of the script. In certain cases you may want more control over this. This setting

suppresses the Date and Symbol columns, so that only your columns are shown.

## 17.12.354. ScanNoHeader

---

### Category

**Settings**

### Description

Allows the header row to optionally be omitted from any **SaveScanAs** output files

### Notes

There is no way to suppress the column headers in a **Scan Window**, but this setting can be used if needed to create a CSV file with no header row.

## 17.12.355. ScanNoWindow

---

### Category

**Settings**

### Description

Allows a **Scan** to create a CSV output file *without* also creating a **Scan Window**.

### Notes

The purpose of this option is to allow large-scale data-generation scans to be run more quickly and with less memory usage.

**SaveScanAs** must be specified to use this setting.

The **Sort** definition, if specified, is ignored in this scan mode.

## 17.12.356. ScanSettings

---

### Category

**Script Sections**

### Description

Defines the settings to use only when the script **run mode** is **Scan**.

### Notes

The general-purpose **Settings** section is always applied first, then modified by any items specified in *ScanSettings* when applicable.

See **Settings Sections** for details.

## 17.12.357. Select

---

### Category

**General-Purpose Functions**

### Description

Conditional choice function

### Syntax

Select(condition, if\_true, condition, if\_true, ...)

### Parameters

condition - formula specifying a true/false condition (non-zero means true)

if\_true - formula to evaluate and return the result of if condition is true

### Notes

Select is similar to **IF** in that it provides a kind of conditional branch function.

In fact, the following two statements are equivalent:

- *if(cond1, value1, value2)*
- *select(cond1, value1, 1, value2)*

The purpose of *select* becomes more apparent when multiple *if* statements need to be nested:

- *if(cond1, if(cond2, if(cond3, value3, NaN), value2), value1)*
- *select(cond1, value1, cond2, value2, cond3, value3)*

In this example, the *select* expression is clearly simpler.

If *select* finds no conditions that return non-zero ("true"), then the function returns NaN. This is different from *if*, which will never return NaN unless done explicitly as in the above example.

To make *select* always return a default value if no conditions match (as in the first *select* example above), either make the final condition simply "1" (or any non-zero value), or use an odd number of arguments:

- *select(cond1, value1, value2)*

When an odd number of arguments is used, the last argument becomes the default value.

## 17.12.358. Sequence

---

### Category

**Multi-Bar Functions**

### Description

Look for a sequence of conditions within a specified number of bars

### Syntax

Sequence(count, condition1, condition2, ...)

### Parameters

count - the count of bars to look within

condition1 - the first condition

condition2 - the second condition

... - any number of additional conditions

### Notes

Returns 1 (true) if condition1 and condition2 and any other conditions specified all occurred within the most recent *count* bars and occurred in the specified sequence.

Returns 0 (false) otherwise.

It does not matter how many bars separate the condition occurrences nor what happens during those intervening bars, only that these conditions occurred in this sequence within this many bars.

### Example

Look for a breakout to a new 100-day high followed by a pullback lasting at least 5 bars followed by

another new high, all within the past 10 bars:

```
▼ Data:
newhigh: C > Highest(H, 100)[1]
breakout: Sequence(10, newhigh, sincetrue(newhigh) >= 5, newhigh)
```

## 17.12.359. Settings

---

### Category

**Script Sections**

### Description

Defines the settings to apply each time the script runs.

### Notes

Use the *Settings* section for all of your common settings.

Use **ScanSettings**, **TestSettings**, and **OrderSettings** as needed to override common settings for these specific **run modes**.

See **Settings Sections** for details.

## 17.12.360. SetupRank

---

### Category

**Current Position Information**

### Description

Returns the rank number for this position when **SetupScore** was evaluated at entry time.

### Notes

*SetupRank* can be referred to in any non-entry-related strategy formula, in **Quantity** if **Side** is specified, and in **EntrySkip**.

The setup ranking mechanism can be observed by running a test with *TestOutput: Log* enabled.

## 17.12.361. SetupScore

---

### Category

**Strategy Elements**

### Description

Ranks entry setups when a strategy has more setups than can be entered

### Input

Numeric formula

### Notes

Setups with higher scores are entered first.

If *SetupScore* is not specified, setups will be ranked in alphabetical order by symbol.

The number of positions that a strategy can enter per day is determined by evaluating the **MaxSetups**, **MaxEntries**, **MaxInvested** and, **MaxPositions**.

For more information on how the backtest engine works in general, see [Backtest Engine Details](#).

## 17.12.362. SetupSkip

---

### Category

**Strategy Elements**

### Description

Enables skipping a setup if a condition applies

### Input

Any formula specifying a true/false condition (non-zero means true)

### Notes

*SetupSkip* is evaluated during the top-down setup selection process. If the return value is *true* (non-zero) then this setup does not become an order and its slot is made available to another setup.

This is in contrast to **EntrySkip**, which is not evaluated until the entry simulation phase of a backtest.

See [BackTest Engine Details](#) for more information about how setup selection works.

## 17.12.363. SetupSum

---

### Category

**General-Purpose Functions**

### Description

Evaluates an expression for each setup and returns the sum of the values

### Syntax

SetupSum(expression)

### Notes

This is a specialized function meant to be used in entry-related formulas that want to the sum of something across all current setups.

*SetupSum(1)* returns the count of today's setups, which is not yet available in **S.Setups** at position entry time.

Use **OrderSum** to count or calculate something for non-skipped setups only.

Note that *SetupSum* cannot be used to refer to past setups in a strategy. It only has access to setups for the current date in the test. Therefore *SetupSum(expression)[offset]* will probably not have meaningful results.

## 17.12.364. Shares or Contracts

---

### Category

**Current Position Information**

### Description

Returns the number of shares or contracts held in the current position

### Notes

This element can be referred to as either *Shares* or *Contracts*, regardless of the type of instrument

being traded.

The number returned is the number of shares (or contracts) held in the current position for the current strategy only.

To get the number of shares of the current symbol held in a different strategy, use **Extern**(*strategy\_name*, *Shares*).

To get the number of shares of the current symbol held in all strategies, use **Combined**(*Shares*).

*Shares* is always a positive number, regardless of the side (long vs. short) of the position.

Multiply *Shares* by **Side** if you need the sign of the result to reflect the position's side.

To get the net long-short position of the current symbol across all strategies use *Combined*(*Shares* \* *Side*).

A common usage of *Shares* is in **Commission** calculations such as *Max*(1.0, 0.005 \* *Shares*).

## 17.12.365. Side (position)

---

### Category

**Current Position Information**

### Description

Position side

### Notes

Returns 1 for a long position or -1 for a short position.

Multiply **Shares** or **FillValue** by *Side* if you want them to be negative for short positions.

## 17.12.366. Side (strategy)

---

### Category

**Strategy Elements**

### Description

Defines the side (long vs. short) on which a strategy will take positions

### Choices

*Long* - all entries are long buys

*Short* - all entries are short sales

*Both* - entries can be either long or short (default)

### Notes

In most cases, it is best to use side-specific strategies when modeling a multi-strategy trading system.

Both-way strategies are mainly useful for special needs such as hedging a long/short strategy pair using an index.

When *Side* is not specified, the sign of the value of **Quantity** is used to determine the side of each entry.

## 17.12.367. Sign

---

## Category

### General-Purpose Functions

## Description

Returns the sign of a number

## Syntax

Sign(value)

## Parameters

value - formula specifying a number

## Notes

Returns 1 if the number is positive, -1 if the number is negative, or 0 if the number is 0.

## 17.12.368. SinceHigh

---

## Category

### Multi-Bar Functions

## Description

Count of bars since the highest value of an expression in a number of bars

## Syntax

SinceHigh(expr, count)

## Parameters

expr - data series formula

count - lookback period

## Notes

If today's value is the highest (largest) value in *count* days, the function returns 0. If the high was yesterday, the return value is 1, and so on.

If there are multiple instances of the highest value then the most recent one is used.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.369. SinceLow

---

## Category

### Multi-Bar Functions

## Description

Count of bars since the lowest value of an expression in a number of bars

## Syntax

SinceLow(expr, count)

## Parameters

expr - data series formula

count - lookback period

## Notes

If today's value is the lowest (smallest) value in *count* days, the function returns 0. If the high was yesterday, the return value is 1, and so on.

If there are multiple instances of the lowest value then the most recent one is used.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

## 17.12.370. SinceTrue

---

### Category

#### Multi-Bar Functions

### Description

Count of bars since a condition was last true

### Syntax

SinceTrue(condition, count {0})

### Parameters

condition - data series formula

count - lookback period (optional)

### Notes

*Condition* will be evaluated for the most recent bar first, then proceed back in time until a non-zero value is found or *count* bars have been checked, whichever comes first.

For each bar, *condition* is evaluated as if that bar were the current bar, i.e. without knowledge of *future* splits relative to that bar.

If *count* is omitted then there is no maximum (all bars before this one are potentially checked).

If *condition* was never true, the return value is -1.

If *condition* is currently true, the return value is 0.

If *condition* was most recently true yesterday, the return value is 1, and so on.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without a count argument.

## 17.12.371. Sine

---

### Category

#### General-Purpose Functions

### Description

Trigonometric sine of a number of degrees

### Syntax

Sine(value)

### Parameters

value - formula

### Notes

The parameter value is assumed to be degrees (0-360).

To convert radians to degrees, multiply by 57.2957795131 (180/n).

## 17.12.372. Skewness

---

### Category

**Multi-Bar Functions**

### Description

Statistical measure of the lopsidedness of a distribution of values

### Syntax

Skewness(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

*Skewness* is calculated in the same way that Excel calculates the SKEW.P function, which is as if the set of *count* values is the entire population.

The specific formula used is shown below, in the "skew" item:

#### ▼ Data:

```
expr: roc(c,1)
mean: avg(expr, count)
sdev: sqr(sum((expr - this(mean)) ^ 2, count) / count)
skew: (1 / count) * (sum((expr - this(mean)) ^ 3, count) / sdev ^ 3)
kurt: (1 / count) * (sum((expr - this(mean)) ^ 4, count) / sdev ^ 4) - 3
```

This also illustrates how these statistical functions could be calculated in the **Data Section** of a script, though since they're provided built-in, there's no reason to do so.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **StdDev** and **Kurtosis**.

## 17.12.373. SkipTestIf

---

### Category

**Settings**

### Description

Formula with condition which, if true, causes an entire test to be skipped

### Notes

This special formula in *TestSettings* is meant to be used when running **optimizations**.

As an example, if you were optimizing a simple RSI or Stochastics crossover strategy, you might use something like this:

#### ▼ Parameters:

```
oversold: from 10 to 90 step 10
overbought: from 10 to 90 step 10
```

#### ▼ TestSettings:

```
SkipTestIf: oversold > overbought
```

Without the *SkipTestIf* filter, there would be 81 tests, including many "illogical" combinations (albeit

perhaps interesting to test these...)

Adding the filter as shown above reduces the actual test count to 45 by eliminating the tests that don't "make sense" for this strategy idea.

This kind of filtering can help large optimizations run significantly faster.

Note that in the context of this formula there is specific strategy or stock. It is only evaluated once per test, before the run begins. The only meaningful elements to use in the formula will be your **Parameters** values.

Note also that the test count shown in the optimization dialog and in the status bar when the test runs indicates the full combination count without the filter. The filtered tests are simply skipped over and the final row count in the results window will be the count of non-skipped tests.

## 17.12.374. Slippage

---

### Category

#### Strategy Elements

### Description

Slippage amount, in points (dollars per share or contract), for each transaction

### Input

Any formula specifying dollars per share or contract (points)

### Notes

Defines the amount of slippage to apply to to each transaction, in price points.

*Slippage* is calculated and applied to each side of the trade (entry and exit) separately.

If **FillPrice** is used in the Slippage formula, it automatically retrieves the entry price for entry slippage and the exit price for exit slippage calculation.

*Slippage*, if specified, is applied by default to every transaction, regardless of order type (market, stop or limit) or time (open, intraday or close).

To specify a different slippage amount for limit-order transactions (fills at a limit price), use **LimitSlip**.

To specify a different slippage amount for stop-order transactions (fills at a stop price) use **StopSlip**.

To specify a different slippage amount for at-open market transactions use **OpenSlip**.

To specify a different slippage amount for at-close market transactions use **CloseSlip**.

If all four of these specific slippage types are specified then there is no need to also specify *Slippage*.

Use *Slippage* when a simple average assumption is sufficient for your modeling needs.

---

For futures, a simple assumption is "one tick" slippage for any market, e.g. Slippage: **TickSize**.

For stocks, percent-based slippage usually makes the most sense, e.g. Slippage:  $0.002 * \text{FillPrice}$ .

A more complex example for less liquid futures, would be:  $\text{Max}(2 * \text{TickSize}, 0.02 * \text{ATR}(5))$ , meaning "2 ticks or 2% of daily average true range, whichever is greater".

A more complex example for less liquid stocks would be:  $\text{Max}(0.05, 0.002 * \text{FillPrice}, 0.1 * \text{Shares} / \text{Volume})$ , meaning "0.05/share or 0.2% of price or 10% of the ratio of your shares to the total daily volume, whichever is greater".

## 17.12.375. Slope

---

## Category

### Multi-Bar Functions

## Description

Linear regression slope

## Syntax

Slope(expr, {expr2,} count)

## Parameters

expr - data series formula (Y values)

expr2 - optional second data series formula (X values -- a linear series from 1 to *count* is used if omitted)

count - lookback period

## Notes

Calculates the slope of a linear regression of *expr* evaluated for the previous *count* bars.

See also [LinReg](#) and [YInt](#).

## 17.12.376. Spearman

---

## Category

### Multi-Bar Functions

## Description

Spearman Rank Correlation of two series

## Syntax

Spearman(expr1, expr2, count)

## Parameters

expr1 - data series formula

expr2 - data series formula

count - lookback period

## Notes

See [this link](#) for a general description of this function.

It was added to RealTest to enable calculation of **this indicator** specifically, i.e.,  $100 * Spearman(C, Barnum, 10)$ .

Perhaps you will find other creative uses for it!

## 17.12.377. Split

---

## Category

### Bar Data Values

## Description

Current bar split factor

## Notes

The split factor of a past bar is defined as the as-traded price divided by the split-adjusted price (unadjusted/adjusted).

If no stock splits have occurred since the date of the past bar, the split factor is therefore 1.0.

Therefore, if *split* is not 1.0, it means there will be splits in the future relative to the current bar being evaluated.

Since RealTest always provides price or volume values "as-traded" (unadjusted), and multi-bar indicators adjusted for past splits (to avoid distortion) but not future ones, there is rarely a need to explicitly refer to a bar's split factor.

Care must be taken to avoid implicit look-ahead bias when using *split* because, by definition, if *split* is not 1.0, this indicates a future split, not a past one.

See also **Split Handling**.

## 17.12.378. Sqr

---

### Category

#### General-Purpose Functions

### Description

Square Root of a number

### Syntax

Sqr(value)

### Parameters

value - formula specifying a numeric value

## 17.12.379. SS

---

### Category

#### Indicator Functions

### Description

Adam Grimes' Sigma Spike indicator

### Syntax

SS(len)

### Parameters

len - lookback period

### Notes

**Sigma Spike** is defined as  $\text{PctChg}(C,1) / \text{StdDev}(\text{PctChg}(C,1),\text{len})[1]$ .

## 17.12.380. StartDate

---

### Category

#### Import Specification

## Description

The first date to include in a data import

## Choices

**Date Constant** - a literal date

*Earliest* - 1/1/1990 or the oldest available date, which ever is more recent

## Notes

To start an import with a date earlier than 1/1/1990, use the desired date rather than *Earliest*. (The actual earliest data date can't be known until the import is completed.)

See also **EndDate** and **NumBars**.

## 17.12.381. StartDate

---

### Category

**Settings**

### Description

The first date to include in a scan or test

### Choices

**Date Constant** - a literal date

*Earliest* - always use the oldest available date

### Notes

If a date range is not specified in a script then the dates from the **Settings Panel** will be used.

## 17.12.382. StartPercent

---

### Category

**Strategy Elements**

### Description

Percent of initial **AccountSize** to give to this strategy

### Input

Any formula specifying a percentage

### Notes

Both **S.StartEquity** and the initial value of **S.Equity** for the strategy will be this percentage of the initial account size.

The default *StartPercent* is 100. This models each strategy having the entire account at its disposal.

Specifying *StartPercent* less than 100 is useful when modeling strategies running in separate accounts. In this case *\*AccountSize\** would be the sum of your initial account values and *StartPercent* would specify the relative percent sizes of each sub-account.

When using *StartPercent: 100* (or not using it since this is the default), multiple strategies should apply their desired allocation fractions via their **Quantity** formulas.

## 17.12.383. StatsGroup

---

## Category

### Script Sections

## Description

Defines a group of strategies to combine for test statistics reporting and for application of multi-strategy constraints such as **MaxExposure**, **MaxInvested**, **MaxPositions**, etc.

## Notes

Group-level capacity constraints are ignored in **Legacy Mode**.

See **Special Strategy Types** for details.

---

## 17.12.384. StdDev

## Category

### Multi-Bar Functions

## Description

Statistical measure of the variance of a distribution of values

## Syntax

StdDev(expr, count)

## Parameters

expr - data series formula

count - lookback period

## Notes

*StdDev* is calculated in the same way that Excel calculates the STDEV.P function, which is as if the set of *count* values is the entire population.

The specific formula used is shown below, in the "sdev" item:

```
▼ Data:
expr: roc(c,1)
mean: avg(expr, count)
sdev: sqr(sum((expr - this(mean)) ^ 2, count) / count)
skew: (1 / count) * (sum((expr - this(mean)) ^ 3, count) / sdev ^ 3)
kurt: (1 / count) * (sum((expr - this(mean)) ^ 4, count) / sdev ^ 4) - 3
```

This also illustrates how these statistical functions could be calculated in the **Data Section** of a script, though since they're provided built-in, there's no reason to do so.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **Skewness** and **Kurtosis**.

---

## 17.12.385. StdErr

## Category

### Multi-Bar Functions

## Description

Standard error of predicted vs. actual values in a regression

## Syntax

StdErr(expr, {expr2,} count)

## Parameters

expr - data series formula (Y values)

expr2 - optional second data series formula (X values -- a linear series from 1 to *count* is used if omitted)

count - lookback period

## Notes

For each value in the *expr* data series formula, the **Slope** and **YInt** are calculated for *count* values. Each *expr* value is then compared to the predicted value of that point on the line thus defined.

The StdError is the square root of the sum of squared differences between actual *expr* values and **LinReg** predictions.

This function is equivalent to =STEYX in Excel.

## Example



## 17.12.386. STOC

### Category

#### Indicator Functions

## Description

Stochastics Indicator

## Syntax

STOC(len, avg)

## Parameters

len - lookback period

avg - smoothing average period

## Notes

The **Stochastic Oscillator** measures the position of a value in a range of similar values, with optional smoothing.

STOC(len, avg) could also be expressed as  $100 * Avg((C - Lowest(L, len)) / (Highest(H, len) - Lowest(L, len)), avg)$ .

STOC(1,1) is therefore a simple way to calculate "Internal Bar Strength", i.e.,  $100*(C-L)/(H-L)$ .

In case you're looking for the "Williams %R" indicator, that is simply  $-1 * (100 - STOC(len, 1))$ .

## 17.12.387. StatsIncludeCash

---

### Category

**Settings**

### Description

Whether to include cash deposits and withdrawals (from **CashInOut** or **CashList**) in percent-based test stats

### Choices

*False* - exclude cash from stats (stats are "time weighted")

*True* - include cash in stats (stats are "net of cash")

### Notes

Use *False* (the default) to see your "pure" trading stats independent of cash withdrawals or deposits (e.g. for taxes).

Use *True* to see trading stats that treat cash transactions like trading gains or losses (e.g. for client "net of fees" stats).

The **management\_fees.rts** example script can be used to apply and report management fees by simply including it at the end of any other script.

## 17.12.388. Strategy

---

### Category

**Script Sections**

### Description

Defines a trading strategy

### Notes

See **Strategy Section** for details.

## 17.12.389. StrategyScore

---

### Category

#### Strategy Elements

### Description

Provides a value to use for this strategy when ranking all strategies to determine setup prioritization in top-down mode

### Input

Numeric formula

### Notes

*StrategyScore* is evaluated for each strategy at each turn of the setup selection process.

The strategy with the highest score gets first choice to add its next setup(s) at that step.

If *StrategyScore* is not provided, the default is simply the negative strategy number (first strategy gets top rank, then second, etc.)

Typically all strategies will use the same score formula (use a shared **Template** if so) though this is not a requirement.

Here are some examples of common *StrategyScore* formulas:

- StratNum // highest strategy number (reverse script order)
- -S.Positions // fewest open positions
- -OrderSum(1) // fewest new orders so far today
- ROC(S.Equity,20) // best recent performance (m2m)
- S.DDPct // worst current drawdown

The fewer setup constraints are in place, the more significant this formula becomes. For example if the only constraint is e.g. *Combined: MaxExposure: 100*, then *StrategyScore* will play a large role in deciding which strategy or set of strategies are most likely to receive that exposure.

*StrategyScore* also gains significance when **MaxPerTurn** is used to allow a strategy to select all of its setups on its first turn, or more than one setup per turn.

Use the **multi\_moc\_top\_down.rts** and **oex\_tf\_top\_down.rts** example scripts to experiment with these.

See Also: **Backtest Engine Details** and **Capacity Constraints**

## 17.12.390. StratNum

---

### Category

#### Current Position Information

### Description

Ordinal number of the current strategy in the script

### Notes

This can be useful in a **StrategyScore** formula.

## 17.12.391. StopSlip

---

## Category

### Strategy Elements

## Description

Slippage amount, in points (dollars per share or contract), for each **stop order** transaction

## Input

Any formula specifying dollars per share or contract (points)

## Notes

Defines the amount of slippage to apply to to each stop order transaction, in price points.

*StopSlip* is applied to any transaction that occurs at an **EntryStop** or **ExitStop** price.

If *StopSlip* is not specified then **Slippage** is applied instead.

When a strategy uses both **EntryStop** and **EntryLimit** (enters positions with a stop-limit order) and the fill is at the limit price, **LimitSlip** is applied.

---

## 17.12.392. Sum

## Category

### Multi-Bar Functions

## Description

Sum of values

## Syntax

Sum(expr, count)

## Parameters

expr - data series formula

count - lookback period

## Notes

Evaluates *expr* for each of *count* bars and returns the sum of the values.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** with a non-variable count.

---

## 17.12.393. SumSince

## Category

### Multi-Bar Functions

## Description

Sum of values since a condition was last true, or until it becomes true

## Syntax

SumSince(condition, expression, count {0}, nth {1})

## Parameters

condition - formula to evaluate for each bar until true (non-zero)

expression - formula to evaluate for each non-true condition bar and add to a running sum

count - how many bars back to go (optional)

nth - which instance of condition to use (optional)

### Notes

For each bar until *condition* is true, *expression* is evaluated and added to the sum.

The sum does not include the value of *expression* for the bar where the condition becomes true.

If *condition* is immediately true, the result is 0.

If *nth* is positive, *condition* is evaluated for the most recent bar, then continues back in time until the *nth* non-zero value is found.

If *nth* is negative, *condition* will be evaluated for the most recent bar, then continues forward in time until the *abs(nth)* non-zero value is found.

If *nth* is not specified, the default is 1.

Count must be provided if *nth* is to be provided (use 0 for the default of "all bars")

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without the optional arguments.

## 17.12.394. SumSQ

---

### Category

**Multi-Bar Functions**

### Description

Sum of squared values

### Syntax

SumSQ(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Evaluates *expr* for each of *count* bars and returns the sum of the squares of the values.

## 17.12.395. Symbol

---

### Category

**Stock/Contract Information**

### Description

Numeric code for the symbol of the current security

### Notes

The value returned is the alphabetical ordinal number of the symbol in the currently loaded **DataFile**.

You don't need to know the actual number to use it. Use **Symbol Constants** like **\$MSFT** to

compare specific symbols to the value of this variable.

See also **?Symbol**, which returns the current symbol as a string, and **SymNum**, which allows dynamic symbol lookup.

?Symbol = "MSFT" and Symbol = \$MSFT would both accomplish the same purpose, but it is slightly more efficient to use symbol constants.

Symbol constants also have the advantage of smart auto-complete when entering them.

## 17.12.396. SymChangeList

---

### Category

#### Settings

### Description

Specifies the path to a CSV file containing a list of symbol changes to use when processing imported trade lists

### Notes

This feature is provided as an optional alternative to editing old trade lists themselves.

The default if not specified is *symchanges.csv* in (a) the current OrderClerkFolder or (b) the RealTest installation folder.

It is not necessary to use the *SymChangeList* for trade list symbols which have since been delisted. When using **Norgate Data** with delisted stocks included, RealTest will automatically find the delisted version of such symbols, e.g. *CATM* would be found as *CATM-202105*, assuming that the trade occurred before that date.

The *SymChangeList* file has two required and two optional columns: *OldSymbol*, *NewSymbol*, *ChangeDate*, *Exchange*. It doesn't matter what the columns are called or whether there's a header row.

The *ChangeDate* column is optional. If supplied, RealTest will only apply the symbol change to imported trades which occurred prior to that date.

The *Exchange* column is optional (*ChangeDate* must also be provided but can be left blank). If supplied it overrides the exchange for this symbol when orders are generated.

Here is an example of a *SymChanges.csv* file used for trade lists:

Prior	Symbol	Date
MTBC	CCLD	1/10/2023
CCNC	GDC	1/10/2023
MPAC	MMV	1/5/2023
GLBL	ALTI	1/4/2023
DNAY	TBIO	1/4/2023
MSPR	LIFW	1/3/2023
GMTX	IRON	12/30/2022

Here is an excerpt from the default *ibfutures.csv* file that is included with RealTest:

Norgate	IB	Date	Exchange
6A	AUD		CME
6B	GBP		CME
6C	CAD		CME
6E	EUR		CME
6J	JPY		CME
6L	BRE		CME
6M	MXP		CME
6N	NZD		CME
6R	RUR		CME
6S	CHF		CME
6Z	ZAR		CME
BTC	BRR		CME
CC	CC		NYBOT
CL	CL		NYMEX
CT	CT		NYBOT
DC	DA		CME
DX	DX		NYBOT
EMD	EMD		CME
ES	ES		CME
ETH	ETHUSDRR		CME
GC	GC		COMEX
GD	GSCI		CME

To correctly generate orders for IB futures trades using Norgate futures data, it is necessary to use this list. Edit as needed for other data sources and/or futures markets.

To use this list, either rename it as *SymChanges.csv* or add *SymChangeList: c:\RealTest\ibfutures.csv* to your settings.

## 17.12.397. SymInfoFile

---

### Category

#### Import Specification

### Description

Specifies a symbol information file to be used during data import

### Input

Path to a symbol information file.

### Notes

See **Symbol Information File** for details.

## 17.12.398. SymNum

---

### Category

#### General-Purpose Functions

### Description

Find the number of a given symbol, and/or allow dynamic external symbol usage

### Syntax

SymNum(symbol)

## Parameters

symbol - any formula returning a string or a number

## Return Value

The current symbol number of the specified symbol, or 0 if not found

## Notes

If the current symbol is MSFT, then  $\$MSFT$ , *Symbol*, *SymNum(Symbol)*, *SymNum(?Symbol)*, and *SymNum("MSFT")* will each return the same number.

*SymNum* can also be used as the first argument to **Extern**, for cases where you need to dynamically refer to other symbols.

*Extern(SymNum("MSFT"), C)* is equivalent to *Extern(\$MSFT, C)*, but is less efficient, because "MSFT" will be looked up every time it is processed whereas \$MSFT is only looked up once, at the start of the test or scan.

A couple of possible use cases:

- Test every permutation of a pair-trade concept in a universe of 30 stocks:

```
▼ Parameters:  
  len: 100  
  sym1: from 1 to 30  
  sym2: from 1 to 30  
  
▼ Data:  
  ratio: Extern(SymNum(sym1),C) / Extern(SymNum(sym2),C)  
  ratioMA: MA(ratio, len)  
  signal: if(ratio >= ratioMA, 1, -1) * if(Symbol=SymNum(sym1), 1, if(Symbol=SymNum(sym2), -1, 0))  
  
▼ Strategy: pair_trade  
  Side: Both // sign of quantity determines side  
  EntrySetup: signal <> signal[1]  
  ExitRule: signal <> signal[1]
```

- Dynamically build specific futures symbol strings:

```
▼ Data:  
  year_ahead: Extern(SymNum(Format("ES-#{?}" , year+1, Right(?Symbol,1))), C)
```

## 17.12.399. T.Bars

---

### Category

**Trade Record Items**

### Description

Duration of a trade in bars

### Notes

Entry and exit the same bar has duration of 0.

Trade duration in days depends on the **BarSize** used to run the test.

## 17.12.400. T.CommIn

---

### Category

**Trade Record Items**

## Description

Commission paid for the entry transaction of a trade, in dollars

## 17.12.401. T.CommOut

---

### Category

**Trade Record Items**

### Description

Commission paid for the exit transaction of a trade, in dollars

## 17.12.402. T.DateIn

---

### Category

**Trade Record Items**

### Description

Entry date of a trade as a number in yyyyymmdd format

## 17.12.403. T.DateOut

---

### Category

**Trade Record Items**

### Description

Exit date of a trade as a number in yyyyymmdd format

## 17.12.404. T.Div

---

### Category

**Trade Record Items**

### Description

Dividend amount(s) received or paid over the duration of a trade

## 17.12.405. T.Fraction

---

### Category

**Trade Record Items**

### Description

The fraction of allocation at trade entry time that was used as the initially ordered position size

### Notes

See **FillFraction** for details about how this is calculated.

## 17.12.406. T.FxIn

---

### Category

**Trade Record Items**

### Description

Currency exchange rate on trade entry date

### Notes

This is the ratio of security currency divided by account currency.

The value will always be 1 unless your test was correctly set up for multi-currency strategy modeling and the stock traded has a different base currency than your account's base currency.

See **Currency** and **Testing Multi-Currency Strategies** for details on how this works.

## 17.12.407. T.FxOut

---

### Category

**Trade Record Items**

### Description

Currency exchange rate on trade exit date

### Notes

This is the ratio of security currency divided by account currency.

The value will always be 1 unless your test was correctly set up for multi-currency strategy modeling and the stock traded has a different base currency than your account's base currency.

See **Currency** and **Testing Multi-Currency Strategies** for details on how this works.

## 17.12.408. T.Highest

---

### Category

**Trade Record Items**

### Description

The highest high during a trade

### Notes

If there was a split during the trade, price is expressed using the exit date split factor.

## 17.12.409. T.Lowest

---

### Category

**Trade Record Items**

### Description

The lowest low during a trade

### Notes

If there was a split during the trade, price is expressed using the exit date split factor.

## 17.12.410. T.NetFx

---

### Category

**Trade Record Items**

### Description

Net currency exchange rate change impact on trade profit or loss

### Notes

This returns the portion of this trade's net profit or loss that is attributable to the change in currency exchange rate between position entry and exit dates.

The value will always be 0 unless your test was correctly set up for multi-currency strategy modeling and the stock traded has a different base currency than your account's base currency.

See **Currency** and **Testing Multi-Currency Strategies** for details on how this works.

## 17.12.411. T.NetPct

---

### Category

**Trade Record Items**

### Description

The net trade profit or loss of a trade after commission and dividend are applied, expressed as a fraction (%/100) of the entry position size

### Notes

This is calculated as  $T.Profit / (pTrade->nShares1 * pTrade->nPrice1 * pTrade->nFxRatio1) / pTrade->nPtVal$ .

See also **T.Profit**

## 17.12.412. T.Points

---

### Category

**Trade Record Items**

### Description

The net points (split-adjusted \$/share) gained or lost in a trade (negative for a loss)

## 17.12.413. T.PriceIn

---

### Category

## Trade Record Items

### Description

Trade entry price

## 17.12.414. T.PriceOut

---

### Category

#### Trade Record Items

### Description

Trade exit price

## 17.12.415. T.Profit

---

### Category

#### Trade Record Items

### Description

The net trade profit or loss of a trade after commission and dividend are applied, expressed in dollars

### Notes

See also **T.NetPct**

## 17.12.416. T.PtVal

---

### Category

#### Trade Record Items

### Description

The **point value** of symbol of a trade

## 17.12.417. T.QtyIn

---

### Category

#### Trade Record Items

### Description

The number of shares or contracts bought or shorted when a trade was entered

## 17.12.418. T.QtyOut

---

## Category

### Trade Record Items

## Description

The number of shares or contracts sold or covered when a trade was exited

## 17.12.419. T.Reason

---

## Category

### Trade Record Items

## Description

The reason a trade was exited or an entry was skipped

## Notes

Trade reason codes are displayed as text in the **trade list window**.

This element is here in case you need to refer to a trade's reason code in a formula.

Exit Reason Codes:

Code	Reason
1	exit rule
2	exit limit
3	exit stop
4	end of test
5	end of data
6	trade list

Skip Reason Codes:

Code	Reason
1	max invested
2	max positions
3	max entries
4	max same category
5	position open
6	zero quantity
7	limit not hit
8	stop not hit
9	ambiguous timing
10	entry skip formula
11	max setups
12	max exposure
13	max new invested
14	stop+limit not hit
15	no data

## 17.12.420. T.Side

---

## Category

## Trade Record Items

### Description

The side of a trade (1=long, -1=short)

## 17.12.421. T.SlipIn

---

### Category

#### Trade Record Items

### Description

The amount of slippage applied to the entry price of a trade, in total dollars

## 17.12.422. T.SlipOut

---

### Category

#### Trade Record Items

### Description

The amount of slippage applied to the exit price of a trade, in total dollars

## 17.12.423. T.SplitIn

---

### Category

#### Trade Record Items

### Description

The bar split factor (actual price / adjusted price) at trade entry time

## 17.12.424. T.SplitOut

---

### Category

#### Trade Record Items

### Description

The bar split factor (actual price / adjusted price) at trade exit time

## 17.12.425. T.Strat

---

### Category

#### Trade Record Items

### Description

The strategy number of a trade

### Notes

Strategies are numbered by the order in which they appeared in a script that was used to run the test

## 17.12.426. T.TimeIn

---

### Category

**Trade Record Items**

### Description

Entry time-of-day code of a trade

### Notes

Trade time codes are displayed as text in the **trade list window**.

This element is here in case you need to refer to a trade's entry time in a formula.

Time codes are: 1=Open, 2=Intraday, 3=Close.

## 17.12.427. T.TimeOut

---

### Category

**Trade Record Items**

### Description

Exit time-of-day code of a trade

### Notes

Trade time codes are displayed as text in the **trade list window**.

This element is here in case you need to refer to a trade's exit time in a formula.

Time codes are: 1=Open, 2=Intraday, 3=Close.

## 17.12.428. T.ValueIn

---

### Category

**Trade Record Items**

### Description

Value calculated by **EntryTradeValue** when position was entered

### Notes

This item can be useful in **Trade Statistics Functions**

## 17.12.429. T.ValueOut

---

### Category

**Trade Record Items**

### Description

Value calculated by **ExitTradeValue** when position was exited

## Notes

This item can be useful in [Trade Statistics Functions](#)

## 17.12.430. Tangent

---

### Category

[General-Purpose Functions](#)

### Description

Trigonometric tangent of a number of degrees

### Syntax

Tangent(value)

### Parameters

value - formula

### Notes

The parameter value is assumed to be degrees (0-360).

To convert radians to degrees, multiply by 57.2957795131 (180/π).

## 17.12.431. Template

---

### Category

[Script Sections](#)

### Description

Defines a strategy template

### Notes

See [Special Strategy Types](#) for details.

## 17.12.432. TargetPrice

---

### Category

[General-Purpose Functions](#)

### Description

Calculate the value of tomorrow's close that would cause an indicator to reach a specific level

### Syntax

TargetPrice(expression, value, range {50})

### Parameters

expression - an indicator formula

value - the desired return value of the indicator

range {50} - today's close plus/minus this percentage sets the range of prices to try

### Return Value

The price that, if it were tomorrow's close, would come nearest to producing the desired indicator value.

## Notes

This function uses binary search logic to govern an iterative process:

- temporarily coerce tomorrow's OHLC values to a specific value in the data (adding an extra bar at the end if needed)
- call the indicator function with a one-bar lookahead offset e.g.  $MA(C, 20)[-1]$
- narrow the search range based on whether the result was too low or too high

The optional *range* argument lets you control how wide a range to allow. For example if today's close was 100 and the default 50% range is used, it will start with the range 50 to 150 and narrow it from there. If the required price is outside the range, the result will be the range boundary, not the correct price.

For a quick test of how this works, compare  $TargetPrice(RSI(2), 50)$  to  $RRSI(2, 50)$  -- the results will be the same (or very close).

(RRSI is much faster than *TargetPrice* because it can calculate the needed price deterministically. *TargetPrice* is meant for use with other indicators that don't support a deterministic reverse version, e.g. **CRSI**.)

The expression passed to *TargetPrice* should not be a reference to a **Data** array containing a previously-calculated indicator. The iterative search process requires multiple recalculations of the indicator while searching for the needed price. *Data* items are not recalculated for this purpose.

It is NOT recommended to use *TargetPrice* within a **Data Section** formula. It is best suited for use in an **EntryLimit** or **ExitLimit** formula, where it will only be evaluated for each setup or open position respectively, not for every bar of every stock in the universe. This is not a speedy function, especially when used with long-lookback indicators.

---

## 17.12.433. TestInclude

### Category

**Script Sections**

### Description

Allows a script to include another script when run in **Test Mode**

### Syntax

*TestInclude*: *path* where *path* is either a full file path (e.g.  $C:\RealTest\Scripts\script.rts$ ) or a path relative to the Scripts folder (e.g.  $Examples\script.rts$ ).

### Notes

See the general-purpose **Include** statement for further details.

---

## 17.12.434. TestData

### Category

**Script Sections**

### Description

Named formulas calculated and stored in memory arrays one bar at a time while tests run

### Notes

See **TestData Section** for a more detailed description.

---

## 17.12.435. TestName

## Category

### Settings

## Description

String that should appear in the "Name" column of the **Test Results** row after the test is run

## Notes

If *TestName* is not specified in a script then the name from the **Settings Panel** will be used, and if that is blank then the script name is used as the test name.

If *TestName* is specified, it is evaluated as a **Formula** that returns a **String Value**. To specify a literal test name, it must be enclosed in quotation marks. **String Functions** such as **Format** can optionally be used to generate a test name that is relevant to the current set of **Parameters**.

## 17.12.436. TestOutput

---

## Category

### Settings

## Description

Specifies additional output and actions during and after a single test run

## Choices (multiple, separated by commas)

*None* - no additional test output

*Report* - generate a **Test Summary Report**

*Graph* - open a **stats graph** and dynamically update it as the test runs

*Log* - generates a detailed **transaction and position log** and displays it at the end of the test

*Orders* - generate the list of **Tomorrow's Orders**

*Scan* - run a **special scan** allowing position-level formula elements to be used to output the list of positions remaining open and/or other details

*Debug* - break into the **debug panel** before closing end-of-test positions to allow full examination of the test context on the last date, and specify whether to show output from **DebugEntry**, **DebugExit** and **DebugTargetStop** statements in the script.

## Notes

If *TestOutput* is not specified in a script then the choices from the **Settings Panel** will remain unchanged and be used.

If *TestOutput: None* is specified then all the check boxes in the Settings Panel will be cleared.

Otherwise, all the check boxes are reset to match the choices specified in this statement.

## 17.12.437. TestScan

---

## Category

### Script Sections

## Description

Test Output Scan definition

## Notes

This section defines a special type of scan that is run at the end of a test and has full access to the

test context including open positions.  
See **Test Output Scan** for more information.

## 17.12.438. TestScanAllDates

---

### Category

**Settings**

### Description

Allows a TestScan to include a row for every date of a test

### Choices

*True* - include all dates

*False* - (default) only run the scan on the last date of the test (before doing end-of-test exits)

### Notes

See **Test Output Scan** for more information.

## 17.12.439. TestSettings

---

### Category

**Script Sections**

### Description

Defines the settings to use only when the script **run mode** is **Test or Optimize**.

### Notes

The general-purpose **Settings** section is always applied first, then modified by any items specified in *TestSettings* when applicable.

See **Settings Sections** for details.

## 17.12.440. This

---

### Category

**General-Purpose Functions**

### Description

Anchors a value in multi-bar or multi-position function calculations

### Syntax

This(value)

### Parameters

value - the value to anchor

### Notes

In functions such as **CountTrue**, **SinceTrue** and **TrueInRow**, we sometimes want to compare a number of prior bar values to one specific bar value.

For example, say you want to know how many bars it has been since there was more volume than today's volume.

You might try using `SinceTrue(V[1] > V)`, but since this is a **multi-bar function**, the entire expression "`V[1] > V`" is rolled back through time and evaluated for each bar as if it was the current bar. So instead of telling you how many bars since volume exceeded today's volume, it is telling you how many bars since any day's volume was greater than the following day's volume.

To get what you're looking for, you'd instead use `SinceTrue(V > This(V))`. When `SinceTrue` rolls back through time, it now only rolls the first 'V' in the expression, while keeping the 'This(V)' anchored to the current bar.

To further understand how this works, consider a simple price with **offset** `C[n]`. If `n=0`, this is today's close, if `n=1`, yesterday's close, and so on. Now consider that `This(C)[n]` might mean. Since `This` has anchored `C` to today's bar, the `[n]` offset has no effect, so `C` will always be returned regardless of the value of `n`.

The **Debug Panel** is a useful place to study complex functions such as this one.

In addition to the above, `This` can also be used in **PositionSum**, **OrderSum**, and **SetupSum** to compare each other position in the loop to the current one.

## 17.12.441. TickSize

---

### Category

**Stock/Contract Information**

### Description

Futures contract tick size

### Notes

Specifies the smallest possible price change for a futures contract.

Tick sizes are obtained automatically when importing data from **Norgate**.

For CSV futures data import, it would be necessary to provide tick sizes for each contract using a **SymInfo** file.

If not provided, `TickSize` defaults to 0.01 for stocks.

Note that `TickSize` is not used in any internal calculations in `RealTest`. All available decimal places in data bar values used in every calculation.

## 17.12.442. TLAdjusted

---

### Category

**Strategy Elements**

### Description

Specifies whether the prices and quantities of trades in a **TradeList** are split-adjusted

### Choices

`False` - not adjusted, i.e. as-traded (default)

`True` - adjusted

### Notes

`RealTest` assumes that `tradelist` prices are not split-adjusted, as in most cases they will have come from actual trades that occurred in the market.

This setting is mainly for use when exporting a trade list from other software such as `AmiBroker` and then importing it to `RealTest`.

## 17.12.443. TLDateFmt

---

### Category

#### Strategy Elements

### Description

Specifies whether the dates in a **TradeList** are in M/D/Y vs. D/M/Y format

### Choices

*DMY* - dates are D/M/Y

*MDY* - dates are M/D/Y

### Notes

The default if a strategy does not specify *TLDateFormat* is to use the *Date Display Format* setting from the **Program Options Dialog**.

This strategy setting lets you use trade lists with the opposite of your standard date format more easily, and/or use two trade lists with opposite formats in the same script.

**Other date formats** are supported which are not ambiguous and therefore do not require *TLDateFmt* to be specified.

## 17.12.444. TLFields

---

### Category

#### Strategy Elements

### Description

Imported trade list CSV field order specification (comma-separated list)

### Choices

*AssetId* - the stock's Norgate asset ID (overrides *Symbol* if provided)

*Symbol* - the stock symbol

*Strategy* - the strategy name (if present, must match script strategy name)

*Side* - the round-trip trade side (long or 1 for long, short or -1 for short)

*Action* - the transaction type (BUY/SELL) and its presence identifies the file as a transaction list

*DateTime* - this field includes both date and time of a transaction, with a space between them

*Shares* - the quantity of a transaction or of both sides of a round-trip trade

*DateIn* - the date of a transaction or the entry date of a round-trip trade

*TimeIn* - the time of a transaction or the entry time of a round-trip trade

*QtyIn* - the quantity of a transaction or the entry quantity of a round-trip trade

*PriceIn* - the price of a transaction or the entry price of a round-trip trade

*FeesIn* - the commission/fees of a transaction or the entry fees of a round-trip trade

*ValueIn* - the **EntryTradeValue** if RealTest generated the trade list or any user-defined value

*DateOut* - the exit date of a round-trip trade

*TimeOut* - the exit time of a round-trip trade

*QtyOut* - the exit quantity of a round-trip trade

*PriceOut* - the exit price of a round-trip trade

*FeesOut* - the exit fees of a round-trip trade

*ValueOut* - the **ExitTradeValue** if RealTest generated the trade list or any user-defined value

## Notes

The *TLFields* order must be specified for a **TradeList** strategy to work.

If the CSV file includes a header row with column labels, these are simply ignored.

If *TLFields* includes Strategy, then only those trades where the contents of that column match the name of the strategy that contains the *TLFields* statement will be included. If *TLFields* does not include Strategy values, then all trades in the list will be used by this strategy.

Here is an example of how a TradeList strategy might look:

```
▼ Strategy: trades  
  TradeList: Examples\actual_trades.csv  
  TLFields: Symbol,,Side,DateIn,QtyIn,PriceIn,DateOut,QtyOut,PriceOut
```

Note that the field names (like all names in RealTest) are not case-sensitive.

To ignore a column in a trade list file, add an extra comma to the field order list, as shown in the example above.

To ignore the first one or more columns, add extra comma(s) at the start of the list, e.g. ",,symbol,side," etc.

See **Using an Imported Trade List** for a detailed description of this mechanism.

## 17.12.445. TLIgnoreRules

---

### Category

**Strategy Elements**

### Description

Specifies whether strategy formulas can override the details of a **TradeList**

### Choices

*False* - don't ignore strategy rules (default)

*True* - ignore strategy rules

### Notes

Adding *TLIgnoreRules: True* to a strategy with a TradeList causes the trades in that list to always be played back verbatim.

In other words, this makes the **Test run mode** work like **Orders** (without generating them) for a hybrid *TradeList* strategy.

## 17.12.446. TLStratName

---

### Category

**Strategy Elements**

### Description

Specifies which strategy name within a **TradeList** maps to the script strategy that imports the TradeList

### Input

A strategy name

## Notes

By default a strategy that imports trades must have the same name as its trades in the list.

This optional strategy setting allows a strategy of any name to import trades of the specified strategy name.

## 17.12.447. TLTimeShift

---

### Category

**Strategy Elements**

### Description

Specifies the timezone shift, in hours, to apply to trade dates and times in a **TradeList**

### Input

A number of hours.

### Notes

This setting is mainly for use with IB Flex Query output for non-US markets where Date/Time is nevertheless reported in NYC time.

For example, if your ASX trades are shown in NYC time, add *TLTimeShift: 14* to your strategy definition.

## 17.12.448. TLValueIn

---

### Category

**Current Position Information**

### Description

The *ValueIn* value for a position that originated from a **TradeList**

### Notes

*TLValueIn* provides a way for strategy formulas to access an arbitrary value from a trade list.

The intended use case is to test combined strategies without having to merge separate scripts.

This is done by running them separately with **SaveTradesAs** and then combining them by playing back those saved trade lists together.

Typically each separate script would use **EntryTradeValue** and/or **ExitTradeValue** to save information such as **OrderPrice** or **FillFraction**.

The combined playback script can then access these values using *TLValueIn* or **TLValueOut**, typically for use in a **Quantity** override.

See **mr\_sample\_long\_only.rts**, **mr\_sample\_short\_only.rts**, and **mr\_sample\_tradelist.rts** for a complete example of how this works.

## 17.12.449. TLValueOut

---

### Category

**Current Position Information**

### Description

The *ValueOut* value for a position that originated from a **TradeList**

## Notes

*TLValueOut* provides a way for strategy formulas to access an arbitrary value from a trade list.

The intended use case is to test combined strategies without having to merge separate scripts.

This is done by running them separately with **SaveTradesAs** and then combining them by playing back those saved trade lists together.

Typically each separate script would use **EntryTradeValue** and/or **ExitTradeValue** to save information such as **OrderPrice** or **FillFraction**.

The combined playback script can then access these values using **TLValueIn** or *TLValueOut*, typically for use in a **Quantity** override.

See **mr\_sample\_long\_only.rts**, **mr\_sample\_short\_only.rts**, and **mr\_sample\_tradelist.rts** for a complete example of how this works.

## 17.12.450. ToDate

---

### Category

**String Functions**

### Description

Convert a string to a date value

### Syntax

ToDate(string)

### Parameters

string - a **literal string** or **string function** result that can be parsed as a date

### Notes

The following date formats can be parsed by this function:

- `yyyymmdd` (as a string)
- `mm/dd/yy` or `mm/dd/yyyy`, where `mm` and `dd` are either 1 or 2 digits
- `dd-mmm-yy` or `dd-mmm-yyyy`, where `mmm` is Jan, Feb, etc. (not case-sensitive)

If the string is not a valid date, the result will be 0.

## 17.12.451. ToLower

---

### Category

**String Functions**

### Description

Convert a string to all lowercase

### Syntax

ToLower(string)

### Parameters

string - a **literal string** or **string function** result

## 17.12.452. ToNum

---

### Category

#### String Functions

### Description

Convert a string to a numeric value

### Syntax

```
ToNum(string, nth)
```

### Parameters

string - a **literal string** or **string function** result that can be parsed as an integer or decimal number

nth - the instance of a number to find within the string (optional)

### Notes

If nth is not specified then this function only looks for a number at the beginning of the string.

If nth is 1 then the first number found within the string (or at the beginning) is returned.

If nth is 2 the second number found is returned, and so on.

A return value of 0 means either the number found was 0 or no number was found.

## 17.12.453. ToUpper

---

### Category

#### String Functions

### Description

Convert a string to all uppercase

### Syntax

```
ToUpper(string)
```

### Parameters

string - a **literal string** or **string function** result

## 17.12.454. Top

---

### Category

#### General-Purpose Functions

### Description

Use the top N digits of a number to make a new number

### Syntax

```
Top(value, digits)
```

### Parameters

value - the number to get top digits from

digits - the count of digits to get

## Notes

This function was added to make it easier to use different levels of the **TRBC** industry codes, though it can be used with any numeric value.

The TRBC codes are 10-digit numbers from which every pair of digits going from left to right makes the industry designation more specific.

For the *economic sector code*, use `Top(TRBC, 2)`.

For the *business sector code*, use `Top(TRBC, 4)`.

For the *industry group code*, use `Top(TRBC, 6)`.

For the *specific industry code*, use `Top(TRBC, 8)`.

## 17.12.455. TradeList

---

### Category

#### Strategy Elements

### Description

Designates a strategy as being based on an external list of trades and provides the path to that list

### Input

Path to a CSV file containing a list of trades to be used.

### Notes

Strategies that include a *TradeList* must also include a **TLFields** definition.

See **Using an Imported Trade List** for a detailed description of this mechanism.

## 17.12.456. Trades

---

### Category

#### Script Sections

### Description

Backtest trade list column definitions

See **Trades Section** and **Trade List Windows**.

## 17.12.457. TradeStatAvg

---

### Category

#### Trade Statistics Functions

### Description

The average of trade record values for the most recent N trades or for all trades

### Syntax

`TradeStatAvg(value, count, symbol)`

### Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted

symbol - specific symbol to include, or all symbols if omitted

#### Notes

See the link above for important information about this function category.

## 17.12.458. TradeStatMax

---

#### Category

**Trade Statistics Functions**

#### Description

The largest of trade record values for the most recent N trades or for all trades

#### Syntax

TradeStatMax(value, count, symbol)

#### Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted

symbol - specific symbol to include, or all symbols if omitted

#### Notes

See the link above for important information about this function category.

## 17.12.459. TradeStatMin

---

#### Category

**Trade Statistics Functions**

#### Description

The smallest of trade record values for the most recent N trades or for all trades

#### Syntax

TradeStatMin(value, count, symbol)

#### Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted

symbol - specific symbol to include, or all symbols if omitted

#### Notes

See the link above for important information about this function category.

## 17.12.460. TradeStatStdDev

---

#### Category

**Trade Statistics Functions**

#### Description

The standard deviation of trade record values for the most recent N trades or for all trades

### Syntax

TradeStatStdDev(value, count, symbol)

### Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted

symbol - specific symbol to include, or all symbols if omitted

### Notes

See the link above for important information about this function category.

## 17.12.461. TradeStatSum

---

### Category

**Trade Statistics Functions**

### Description

The sum of trade record values for the most recent N trades or for all trades

### Syntax

TradeStatSum(value, count, symbol)

### Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted

symbol - specific symbol to include, or all symbols if omitted

### Notes

See the link above for important information about this function category.

## 17.12.462. Trough

---

### Category

**Multi-Bar Functions**

### Description

Value of the nth most recent trough of a series of prices or other values

### Syntax

Trough(expr, pctChg, nth {1})

### Parameters

expr - data series formula

pctChg - percent change required to delimit peaks and troughs

nth - which trough to locate (1, i.e., most recent if omitted)

### Notes

The definition of a Trough is the lowest value which is then followed by one or more values that are at least n% above that most recent low.

See **Peak** for important additional information about how these functions work in RealTest.

## 17.12.463. TroughBars

---

### Category

#### Multi-Bar Functions

### Description

Count of bars since the nth most recent trough of a series of prices or other values

### Syntax

```
TroughBars(expr, pctChg, nth {1})
```

### Parameters

expr - data series formula

pctChg - percent change required to delimit peaks and troughs

nth - which trough to locate (1, i.e., most recent if omitted)

### Notes

The definition of a Trough is the lowest value which is then followed by one or more values that are at least n% above that most recent low.

See **Peak** for important additional information about how these functions work in RealTest.

## 17.12.464. TrueInRow

---

### Category

#### Multi-Bar Functions

### Description

Count of bars in a row for which a condition was true

### Syntax

```
TrueInRow(expr, count {0})
```

### Parameters

condition - data series formula

count - lookback period (optional)

### Notes

*Condition* will be evaluated for the most recent bar first, then proceed back in time until a FALSE (0) value is found or count bars have been checked, whichever comes first.

For each bar, *condition* is evaluated as if that bar were the current bar, i.e. without knowledge of *future* splits relative to that bar.

If *count* is omitted then there is no maximum (all bars before this one are potentially checked).

If *condition* was never true, the return value is 0.

If *condition* is currently true, but wasn't true yesterday, the return value is 1.

If *condition* was true today and yesterday, the return value is 2, and so on.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without a count argument.

## 17.12.465. TrueRange or TR

---

### Category

#### Bar Data Values

### Description

Current bar range including prior close

### Notes

True Range is defined as  $Max(C[1],H) - Min(C[1],L)$ .

Either *TrueRange* or *TR* can be used.

See also **ATR**, **Range**.

## 17.12.466. UntilTrue

---

### Category

#### Multi-Bar Functions

### Description

Count bars until a condition will be true

### Syntax

UntilTrue(condition, count {0}) {WARNING: looks ahead}

### Parameters

condition - data series formula

count - lookback period (optional)

### Notes

*Condition* will be evaluated for the most recent bar first, then proceed forward in time until a non-zero value is found or count bars have been checked, whichever comes first.

For each bar, *condition* is evaluated as if that bar were the current bar, i.e. without knowledge of *future* splits relative to that bar.

If *count* is omitted then there is no maximum (all bars after this one are potentially checked).

If *condition* never becomes true, the return value is -1.

If *condition* is currently true, the return value is 0.

If *condition* will be true tomorrow, the return value is 1, and so on.

Since this function looks into the future, it should generally not be used with price data in a backtest.

The most likely usage scenario would be date-related, e.g. *UntilTrue(Month <> Month[1])* to count the number of remaining trading days this month.

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without a count argument.

## 17.12.467. Update

---

### Category

#### Import Specification

## Description

Norgate data update request

## Choices

*True* - launch a Norgate update

*False* - don't launch a Norgate update (default)

## Notes

This import option makes it easy to ensure that **Norgate** has updated the data before you import it.

When a Norgate import is run with *Update: TRUE*, the update process is launched and RealTest waits for it to finish before starting the import. The status of this operation is shown in the **Status Bar**. It is fine to be editing scripts, reviewing test results, etc. while an import (or any other task) is running.

## 17.12.468. UseAvailableBars

---

### Category

**Settings**

### Description

Controls what happens when there are not enough bars to calculate a simple average or indicator

### Choices

*True* - calculate simple averages and indicators using how ever many bars are available

*False* - return NaN if not enough bars are available (default)

### Notes

When *UseAvailableBars* is *False*, then any simple average or indicator that does not have enough data to calculate the correct value will return *NaN* (not a number), and the formula will stop being evaluated.

When *UseAvailableBars* is *True*, simple averages and indicators are calculated using however many bars are available. For example, if a formula refers to *Avg(C,200)* and the current bar is only the 100th bar of the data for that symbol, then the 100-day average close will be returned instead.

The *UseAvailableBars* setting has no effect on exponential averages or indicators. Exponential average calculation actually requires five times the "length" to reach full precision. RealTest will use this full-precision length when it is available, and will use however many bars are available otherwise. Unlike simple averages, however, this does not change the "length" parameter, which is really just the denominator of the fraction used to multiply each new value by the prior result during calculation. So for exponential averages, having too few bars available has a less significant impact on the result than it does for simple averages.

See also **Number of Bars Required**.

## 17.12.469. Using

---

### Category

**Strategy Elements**

### Description

Tells a **Strategy** or **Benchmark** or **Template** to use elements from another strategy or benchmark or template

## Input

A list of comma-separated names of strategies or benchmarks of templates

## Notes

*Using* is most often used by strategies to incorporate elements from templates, but any strategy type can inherit from any other with *Using*.

All of the elements defined in the used strategy are copied into the using strategy. If the using strategy defines its own version of any of those elements, its own elements replace the ones from the used strategy.

All this is here just to save you from having to re-type (or copy/paste) strategy elements that you use in multiple strategies in a script. Certain elements, such as *Commission* and *Slippage* formulas, are often the same for every strategy.

Some of the **example scripts**, such as **mr\_sample.rts**, incorporate the template/using concept.

## 17.12.470. Volume or V

---

### Category

**Bar Data Values**

### Description

Current bar volume

## 17.12.471. WalkForward

---

### Category

**Script Sections**

### Description

Defines lists of system parameter values by date for use in running a walk-forward test that was generated by a walk-forward optimization process

### Notes

See **Optimization Dialog** and **Walk-Forward Tests** for details.

If a script includes a *WalkForward* section (not commented out), running it in single test mode will always use the variable parameter values as listed under *WalkForward*. To run such a script with constant parameter values, comment out the *WalkForward* section first.

See *spy\_tlt\_uis.rts* in the *Examples* folder for an example that uses walk-forward optimization to implement a strategy from an article.

## 17.12.472. Week

---

### Category

**Bar Data Values**

### Description

Current bar week of year (1-52)

## 17.12.473. WhenTrue

---

### Category

#### Multi-Bar Functions

### Description

Evaluates an expression for a past (or future) bar when a condition was (or will be) true

### Syntax

WhenTrue(condition, expression, count {0}, nth {1})

### Parameters

condition - formula to evaluate for each bar until true (non-zero)

expression - formula to evaluate when condition is true

count - how many bars back to go (optional)

nth - which instance of condition to use (optional)

### Notes

If *nth* is positive, *condition* is evaluated for the most recent bar and continue back in time until the *nth* non-zero value is found.

If *nth* is negative, *condition* will be evaluated for the most recent bar and continue forward in time until the *abs(nth)* non-zero value is found.

For each bar, *condition* is evaluated as if that bar were the current bar, i.e. without knowledge of *future* splits relative to that bar.

When the specified *nth* condition is found, *expression* is then evaluated for that bar and the resulting value is returned.

If *condition* is never found, the result is NaN.

If *nth* is not specified, the default is 1.

Count must be provided if *nth* is to be provided (use 0 for the default of "all bars").

This function supports ultra-fast **one-pass calculation** when used in the **Data Section** without the optional arguments.

## 17.12.474. WMA or WAvg

---

### Category

#### Multi-Bar Functions

### Description

Weighted Moving Average

### Syntax

WMA(expr, count)

### Parameters

expr - data series formula

count - lookback period

### Notes

Either *WMA* or *WAvg* can be used as the name of this function.

This type of moving average is calculated by putting the most weight on the most recent bar, fractionally less weight on the next bar, and so on. [This link](#) describes how the calculation is done in more detail.

## 17.12.475. Year

---

### Category

#### Bar Data Values

### Description

Current bar year number

Negative offsets, e.g. *Year[-2]*, can be legitimately used to obtain the year of a future bar. This works even if the offset goes beyond the range of the currently loaded data file. For best results when future dates are required, a [HolidayList](#) should also be provided.

## 17.12.476. YInt

---

### Category

#### Multi-Bar Functions

### Description

Linear regression y-intercept

### Syntax

`YInt(expr, {expr2,} count)`

### Parameters

*expr* - data series formula (Y values)

*expr2* - optional second data series formula (X values -- a linear series from 1 to *count* is used if omitted)

*count* - lookback period

### Notes

Calculates the y-intercept of a linear regression of *expr* evaluated for the previous *count* bars.

See also [Slope](#) and [LinReg](#).