RealTest User Guide

Copyright © 2020-2026 Systematic Solutions, LLC

Table of Contents

1. Welcome! **16**
2. Getting Started **17**
2.1. Computer Requirements 17
2.2. File Types 17
2.3. File Paths 18
2.4. Backups 19
3. Brief Tutorials **20**
3.1. Tutorial 1 - SPY Crossover 20
3.2. Tutorial 2 - Simple Optimization 25
3.3. Tutorial 3 - Simple Scan 32
3.4. Tutorial 4 - ETF Rotation 35
4. Classic Example Scripts **41**
5. New Example Scripts **46**
6. Software User Interface **51**
6.1. The Main Window 51
6.2. Common Menus 53
6.2.1. File Menu 53
6.2.2. Edit Menu 54
6.2.3. View Menu 57
6.2.4. Data Menu 57
6.2.5. Run Menu 59
6.2.6. Window Menu 59
6.2.7. Help Menu 60
6.3. Child Window Menus 61
6.3.1. Chart Menu 61
6.3.2. Graph Menu 62
6.3.3. List Menu 64
6.3.4. Log Menu 64
6.3.5. Plot Menu 65
6.3.6. ResGraph Menu 66
6.3.7. Results Menu 68
6.3.8. Scan Menu 70
6.3.9. Script Menu 71
6.3.10. Compare Menu 73
6.3.11. Trades Menu 74
6.4. Bars and Panels 74
6.4.1. Tool Bar 74
6.4.2. Status Bar 76
6.4.3. Files Panel 76
6.4.4. Settings Panel 78
6.4.5. Debug Panel 80
6.5. Text Editor Windows 83
6.5.1. Script Windows 84
6.5.2. Log Windows 85
6.6. List Windows 85
6.6.1. Common Features 86
6.6.2. Results Windows 88
6.6.3. Trade List Windows 89
6.6.4. Scan Windows 92
6.6.5. Trade Comparison Windows 94
6.6.6. Other List Windows 96
6.7. Graphical Windows 96
6.7.1. Daily Stats Graphs 96
6.7.2. Optimization Results Graphs 103
6.7.3. Candlestick/Bar Charts 107
6.7.4. Trade Plots and Analysis 111

6.7.4.1. Factor Scan ... 121
6.8. Program Options Dialog ... 125
6.9. RealTest Function Key Reference ... 133
7. Importing Bar Data ... 135
7.1. How Importing Works ... 135
7.2. Norgate Data Import ... 136
7.3. CSI Futures Import ... 139
7.4. CSI Stocks Import ... 144
7.5. CSV Import ... 148
7.6. EODHD Import ... 149
7.7. MetaStock Import ... 149
7.8. Tiingo Import ... 150
7.9. Yahoo Import ... 150
7.10. Multi-Source Import ... 151
7.11. Symbol Information Files ... 154
7.12. Futures Symbol Information ... 156
7.13. Event List Files ... 156
7.14. Dividend and Constituency Events ... 157
8. Running Scans ... 159
8.1. How Scans Work ... 159
8.2. Data Scan ... 159
8.3. Test Output Scan ... 160
8.4. Multi-Filter Scan ... 161
9. Running Tests ... 162
9.1. How Tests Work ... 162
9.2. Multiple Tests and Optimization ... 162
9.3. Walk-Forward Tests ... 166
9.4. In-Sample / Out-Of-Sample ... 169
9.5. Overfitting Analysis ... 171
10. Analyzing Test Results ... 176
10.1. The Results Window ... 176
10.2. Test Summary Report ... 177
10.3. Test Details Log ... 178
11. Trading Your System ... 180
11.1. Before You Go Live ... 180
11.2. Tomorrow's Orders ... 180
11.3. Generated Order Types ... 183
11.4. OrderClerk ... 184
11.5. Symbol Changes in Open Positions ... 185
11.6. CSV Order Baskets ... 187
11.7. IB Rebalance Tool ... 189
12. Other Ways to Use RealTest ... 191
12.1. Using an Imported Trade List ... 191
12.2. Using Command Line Mode ... 194
12.3. Using the Automation API ... 196
12.4. Using Multiple Instances ... 199
12.5. Using RealTest with Agentic AI ... 200
13. Backtest Engine Details ... 202
13.1. Backtest Loop Overview ... 202
13.2. Asset Allocation and Position Sizing ... 204
13.3. Capacity Constraints ... 205
13.4. Compounding ... 207
13.5. Split Handling ... 210
13.6. Dividend Handling ... 212
13.7. End-of-Data Exits ... 216
13.8. Intraday Fills With Daily Bars ... 217
13.9. Intraday Fill Sequence Assumptions ... 218
13.10. Bar Sizes and Multiple Timeframes ... 219
13.11. Calendar Alignment ... 219

13.12. Calculation of Trade Excursions ... 220
13.13. The Current Bar in Formula Evaluation ... 220
13.14. Specifying a Time Stop ... 221
13.15. Number of Bars Required for Functions and Indicators ... 222
13.16. Scaling In or Out of Positions ... 223
13.17. Referring to Past Trades in Strategy Formulas ... 226
13.18. Testing Multi-Currency Strategies ... 226
13.19. Using CSI Futures Data ... 229
14. RealTest Script Language ... 233
14.1. Language Basics ... 233
14.2. Script Sections ... 235
14.2.1. Import Section ... 236
14.2.2. Data Section ... 236
14.2.2.1. How the Data Section Works ... 236
14.2.2.2. One-Pass Data Formulas ... 237
14.2.2.3. Self-Referential Items ... 239
14.2.2.4. Bar-Size-Specific Items ... 240
14.2.2.5. DataValueFile Items ... 242
14.2.2.6. Data On Demand ... 243
14.2.3. TestData and StratData Sections ... 244
14.2.4. Scan and TestScan Sections ... 248
14.2.5. Settings Sections ... 249
14.2.6. Library Section ... 252
14.2.7. Strategy Section ... 253
14.2.7.1. Strategy Basics ... 253
14.2.7.2. Strategy Names ... 254
14.2.7.3. Special Strategy Types ... 255
14.2.7.4. Strategy Elements ... 256
14.2.7.5. Strategy Element Value Types and Defaults ... 257
14.2.8. Parameters Section ... 259
14.2.9. Results Section ... 261
14.2.10. Graphs Section ... 263
14.2.11. Trades Section ... 265
14.2.12. Charts Section ... 266
14.2.13. Include Section ... 269
14.3. Formula Syntax ... 270
14.3.1. Formula Basics ... 270
14.3.2. Operators ... 271
14.3.3. Formula Evaluation ... 272
14.3.4. Breadth Tags / Cross-Sectional Functions ... 273
14.3.5. Special Syntax for Individual Futures Contract Testing ... 274
14.3.6. Statistics Values in Formulas ... 276
14.4. Bar Offsets ... 277
14.5. Date Constants ... 278
14.6. Other Constants ... 278
14.7. Symbol References ... 279
14.8. String Values ... 279
14.9. User-Defined Item Names ... 280
14.10. Item Name Overrides ... 281
14.11. Script Comments ... 281
14.12. Conditional Comments ... 282
14.13. Repeat Blocks ... 283
14.14. File Path Syntax ... 285
14.15. Output Format Specification ... 286
14.16. New Script Template ... 288
14.17. Combining Scripts ... 288
14.18. Default Scripts ... 290
14.19. Syntax Element Categories ... 291
14.19.1. Script Sections ... 291

14.19.2. Settings ... 292
14.19.3. Optimize Settings ... 295
14.19.4. Scan Settings ... 296
14.19.5. Order Settings ... 296
14.19.6. Import Specification ... 297
14.19.7. Strategy Elements ... 298
14.19.8. Bar Data Values ... 301
14.19.9. Indicator Functions ... 302
14.19.10. Multi-Bar Functions ... 303
14.19.11. Cross-Sectional Functions ... 304
14.19.12. General-Purpose Functions ... 305
14.19.13. String Functions ... 307
14.19.14. Stock/Contract Information ... 307
14.19.15. Current Position Information ... 308
14.19.16. Current Strategy Information ... 309
14.19.17. Test Statistics Arrays ... 309
14.19.18. Trade Record Values ... 311
14.19.19. Trade Statistics Functions ... 313
14.20. Syntax Element Details ... 313
14.20.1. #Avg ... 313
14.20.2. #ByCII ... 314
14.20.3. #ByEcon ... 314
14.20.4. #ByGroup ... 314
14.20.5. #ByIndu ... 315
14.20.6. #ByListNum ... 315
14.20.7. #ByMkt ... 316
14.20.8. #BySect ... 316
14.20.9. #Count ... 316
14.20.10. #DataValueFile ... 317
14.20.11. #DenseRank ... 317
14.20.12. #DVFAlign ... 318
14.20.13. #Fill ... 318
14.20.14. #Highest ... 319
14.20.15. #Lowest ... 319
14.20.16. #Median ... 319
14.20.17. #OnePerDate ... 320
14.20.18. #OnePerSym ... 320
14.20.19. #Pad ... 321
14.20.20. #PercentRank ... 321
14.20.21. #PercentRankN ... 322
14.20.22. #Rank ... 322
14.20.23. #SlowCalc ... 323
14.20.24. #StdDev ... 323
14.20.25. #Sum ... 323
14.20.26. ?AccountSize ... 324
14.20.27. ?CII ... 324
14.20.28. ?Currency ... 324
14.20.29. ?Domicile ... 324
14.20.30. ?EconSect ... 325
14.20.31. ?EndDate ... 325
14.20.32. ?EquityType ... 325
14.20.33. ?Exchange ... 325
14.20.34. ?IndGroup ... 326
14.20.35. ?Industry ... 326
14.20.36. ?ListingType ... 326
14.20.37. ?LocalTime ... 326
14.20.38. ?Name ... 327
14.20.39. ?OrdersDate ... 327
14.20.40. ?ReportingCurrency ... 327

14.20.41. ?RunMode ... 328
14.20.42. ?ScriptName ... 328
14.20.43. ?Sector ... 328
14.20.44. ?Strategy ... 328
14.20.45. ?StratType ... 329
14.20.46. ?StartDate ... 329
14.20.47. ?Symbol ... 329
14.20.48. ?Type ... 330
14.20.49. Abs ... 330
14.20.50. AccountSize ... 330
14.20.51. AccountType ... 331
14.20.52. Adjustment ... 331
14.20.53. Adjusted ... 332
14.20.54. ADX ... 333
14.20.55. AEMA ... 333
14.20.56. AESD ... 334
14.20.57. Allocation ... 334
14.20.58. AllowMissingBar ... 335
14.20.59. AllowNoVolume ... 336
14.20.60. AllowSameName ... 336
14.20.61. AlwaysImport ... 337
14.20.62. Ambiguity ... 337
14.20.63. ArcCos ... 337
14.20.64. ArcSin ... 338
14.20.65. ArcTan ... 338
14.20.66. Arg1-Arg9 / Ref1-Ref9 ... 339
14.20.67. Assert ... 339
14.20.68. ATR ... 340
14.20.69. BarDate ... 341
14.20.70. BarNum ... 341
14.20.71. BarsHeld ... 341
14.20.72. BarsLeft ... 342
14.20.73. BarSize ... 342
14.20.74. BarStart ... 343
14.20.75. BBBot ... 343
14.20.76. BBBotF ... 343
14.20.77. BBPct ... 344
14.20.78. BBPctF ... 345
14.20.79. BBTop ... 345
14.20.80. BBTopF ... 346
14.20.81. BBTrend ... 346
14.20.82. BBTrendF ... 347
14.20.83. BBWidth ... 347
14.20.84. BBWidthF ... 348
14.20.85. Benchmark ... 348
14.20.86. Bound ... 349
14.20.87. CalendarSym ... 349
14.20.88. CashInOut ... 350
14.20.89. CashIntPct ... 350
14.20.90. CashList ... 351
14.20.91. Category (definition) ... 352
14.20.92. Category (reference) ... 352
14.20.93. CCI ... 352
14.20.94. CDF ... 353
14.20.95. Chart Settings ... 353
14.20.96. Charts ... 354
14.20.97. CIIFamily ... 354
14.20.98. CIILevel ... 355
14.20.99. Classification ... 355

14.20.100. Close or C ... 356
14.20.101. CloseSlip ... 356
14.20.102. Collateral / FuturesMargin ... 356
14.20.103. Combined (function) ... 357
14.20.104. Combined (section) ... 358
14.20.105. CommCurrency ... 358
14.20.106. Commission ... 358
14.20.107. Compounded ... 359
14.20.108. Constituency ... 359
14.20.109. Correl ... 361
14.20.110. CorrelAvg ... 361
14.20.111. CorrelMed ... 362
14.20.112. CorrelMin ... 362
14.20.113. CorrelMinSym ... 363
14.20.114. CorrelMax ... 363
14.20.115. CorrelMaxSym ... 364
14.20.116. Cosine ... 364
14.20.117. Cross ... 365
14.20.118. CountTrue ... 365
14.20.119. CRSI ... 366
14.20.120. CSVDateFmt ... 366
14.20.121. CSVDelim ... 367
14.20.122. CSVFields ... 367
14.20.123. CSVFile ... 369
14.20.124. CSVNumFmt ... 369
14.20.125. Currency ... 369
14.20.126. Data ... 370
14.20.127. DataFile ... 370
14.20.128. DataPath ... 370
14.20.129. DataSource ... 371
14.20.130. DataType ... 371
14.20.131. DataValueFile ... 372
14.20.132. Date ... 372
14.20.133. DateBars ... 373
14.20.134. DateDay ... 373
14.20.135. DateDisplay ... 373
14.20.136. DateInput ... 374
14.20.137. DateMonth ... 374
14.20.138. DateSym ... 374
14.20.139. DateYear ... 375
14.20.140. Day ... 375
14.20.141. DayOfWeek ... 375
14.20.142. DayOfYear ... 376
14.20.143. Days ... 376
14.20.144. DaysPerYear ... 377
14.20.145. DividendsToCash ... 377
14.20.146. DebugEntry ... 378
14.20.147. DebugEval ... 378
14.20.148. DebugExit ... 379
14.20.149. DebugTargetStop ... 379
14.20.150. DEMA ... 380
14.20.151. Dividend ... 380
14.20.152. DllDataCalc ... 381
14.20.153. DynamicSizing ... 381
14.20.154. EhlersBP ... 382
14.20.155. EhlersHP ... 383
14.20.156. EhlersSS ... 384
14.20.157. EhlersUS ... 384
14.20.158. EMA or XAvg ... 385

14.20.159. EndDate (Import) ... 386
14.20.160. EndDate (Setting) ... 386
14.20.161. EndOfMonth ... 386
14.20.162. EndOfQuarter ... 387
14.20.163. EndOfTestExits ... 387
14.20.164. EndOfWeek ... 388
14.20.165. EndOfYear ... 388
14.20.166. EntryDate ... 388
14.20.167. EntryLimit ... 389
14.20.168. EntryPhase ... 389
14.20.169. EntryRank ... 390
14.20.170. EntryScore ... 391
14.20.171. EntrySetup ... 391
14.20.172. EntrySkip ... 392
14.20.173. EntryStop ... 392
14.20.174. EntryTime ... 393
14.20.175. EntryTradeValue ... 394
14.20.176. ESD ... 395
14.20.177. Event ... 395
14.20.178. EventListFile ... 396
14.20.179. ExchangeMap ... 396
14.20.180. ExcludeIf ... 399
14.20.181. ExcludeList ... 399
14.20.182. ExitLimit ... 400
14.20.183. ExitLimitQty ... 400
14.20.184. ExitLimitTime ... 401
14.20.185. ExitNum ... 402
14.20.186. ExitQty ... 402
14.20.187. ExitRank ... 403
14.20.188. ExitRule ... 403
14.20.189. ExitScore ... 404
14.20.190. ExitStop ... 404
14.20.191. ExitStopQty ... 405
14.20.192. ExitStopTime ... 405
14.20.193. ExitTime ... 406
14.20.194. ExitTradeValue ... 407
14.20.195. ExitType ... 407
14.20.196. Exp ... 407
14.20.197. Extern ... 408
14.20.198. Extra ... 409
14.20.199. F.xxx / F.xxx.Date ... 409
14.20.200. FeesInOut ... 410
14.20.201. FeesList ... 411
14.20.202. FillFraction ... 411
14.20.203. FillPrice ... 412
14.20.204. FillPriceAvg ... 412
14.20.205. FillPriceMax ... 412
14.20.206. FillPriceMin ... 413
14.20.207. FillQty ... 413
14.20.208. FillValue ... 414
14.20.209. FilterNum ... 414
14.20.210. Find ... 414
14.20.211. Format ... 415
14.20.212. FormatDate ... 415
14.20.213. Fundamentals ... 416
14.20.214. FunBar ... 417
14.20.215. FunStat ... 418
14.20.216. FxRatio ... 418
14.20.217. GetField ... 419

14.20.218. GreedyScoring 419
14.20.219. Graphs 420
14.20.220. High or H 420
14.20.221. Highest or HHV 420
14.20.222. HMA or HAvg 421
14.20.223. HolidayList 421
14.20.224. Hurst 422
14.20.225. HVOL 423
14.20.226. ICDF 423
14.20.227. IF 424
14.20.228. IgnoreDividends 424
14.20.229. Import 425
14.20.230. Include 425
14.20.231. IncludeList 426
14.20.232. InXXX 427
14.20.233. InfoID 427
14.20.234. InfoDelist / InfoExpiry 427
14.20.235. InfoFloat 428
14.20.236. InfoGICS 428
14.20.237. InfoMargin 429
14.20.238. InfoShares 429
14.20.239. InfoTRBC 429
14.20.240. InList 430
14.20.241. InSet 430
14.20.242. IsExit 431
14.20.243. IsNaN 431
14.20.244. IsOrder 433
14.20.245. IsSetup 433
14.20.246. IsTracker 434
14.20.247. Item 434
14.20.248. JDSAR 435
14.20.249. KAMA 436
14.20.250. KBBot 436
14.20.251. KBTop 437
14.20.252. KeepAdjusted 437
14.20.253. KeepNonIncluded 438
14.20.254. KeepRedundant 438
14.20.255. KeepTrades 439
14.20.256. Kurtosis 439
14.20.257. Left 440
14.20.258. Length 440
14.20.259. Library 441
14.20.260. LimitExtra 441
14.20.261. LimitSlip 442
14.20.262. LinReg 442
14.20.263. ListNum 443
14.20.264. Log 443
14.20.265. LogDetail 443
14.20.266. LogFile 444
14.20.267. Logit 445
14.20.268. Low or L 446
14.20.269. Lowest or LLV 446
14.20.270. MA or Avg 446
14.20.271. MACD 447
14.20.272. MACDH 447
14.20.273. MACDS 448
14.20.274. MarginIntPct 448
14.20.275. MarkToMarket 449
14.20.276. Match 449

14.20.277. Max ... 450
14.20.278. MaxN ... 451
14.20.279. MaxCatExp ... 451
14.20.280. MaxCatInv ... 452
14.20.281. MaxEntries ... 452
14.20.282. MaxExposure ... 453
14.20.283. MaxInvested ... 453
14.20.284. MaxLongExp ... 454
14.20.285. MaxLongInv ... 454
14.20.286. MaxNetExp ... 455
14.20.287. MaxNetInv ... 456
14.20.288. MaxNewExp ... 456
14.20.289. MaxNewInv ... 456
14.20.290. MaxNewPos ... 457
14.20.291. MaxPerTurn ... 457
14.20.292. MaxPositions ... 458
14.20.293. MaxSameCat ... 458
14.20.294. MaxSameSym ... 459
14.20.295. MaxSetups ... 459
14.20.296. MaxShortExp ... 459
14.20.297. MaxShortInv ... 460
14.20.298. MaxSymExp ... 461
14.20.299. MaxSymInv ... 461
14.20.300. MDI ... 462
14.20.301. Median ... 462
14.20.302. Mid ... 463
14.20.303. Min ... 463
14.20.304. MinN ... 463
14.20.305. MinFreeCash ... 464
14.20.306. MinNetExp ... 465
14.20.307. MinNetInv ... 465
14.20.308. MissingBars ... 466
14.20.309. Month ... 466
14.20.310. MostRecent ... 467
14.20.311. Namespace ... 468
14.20.312. NextOpen ... 469
14.20.313. NoNan ... 469
14.20.314. Notes ... 470
14.20.315. NoWeekends ... 470
14.20.316. NthOfMonth ... 470
14.20.317. NumBars ... 471
14.20.318. OBV ... 472
14.20.319. Open or O ... 472
14.20.320. OpenSlip ... 472
14.20.321. OptAnchor ... 473
14.20.322. OptByStrategy ... 473
14.20.323. OptBySymbol ... 473
14.20.324. OptClearResults ... 474
14.20.325. OptContinueResults ... 474
14.20.326. OptimizeMode ... 474
14.20.327. OptimizeSettings ... 475
14.20.328. OptIterations ... 475
14.20.329. OptKeepBest ... 476
14.20.330. OptKeepByDate ... 476
14.20.331. OptKeepBySymbol ... 476
14.20.332. OptKeepNoTrades ... 476
14.20.333. OptMultiSaveAs ... 477
14.20.334. OptMutatePct ... 477
14.20.335. OptNoDialog ... 477

14.20.336. OptNoStats ... 477
14.20.337. OptOverfit ... 478
14.20.338. OptOverfitBlocks ... 478
14.20.339. OptScoreCol ... 478
14.20.340. OptShards ... 479
14.20.341. OptShowResults ... 479
14.20.342. OptSortResults ... 479
14.20.343. OptSummaryLog ... 480
14.20.344. OptTestInterval ... 480
14.20.345. OptTestLength ... 480
14.20.346. OptTimeUnit ... 480
14.20.347. OptWalkForward ... 481
14.20.348. OrderClerkFolder ... 481
14.20.349. OrderNote / OrderExtra ... 481
14.20.350. OrderPrice ... 483
14.20.351. OrderRank ... 483
14.20.352. OrdersClassDelim ... 484
14.20.353. OrdersComment ... 484
14.20.354. OrderSettings ... 485
14.20.355. OrdersFile ... 485
14.20.356. OrdersFreeCash ... 485
14.20.357. OrdersIgnoreSplits ... 486
14.20.358. OrdersInclude ... 486
14.20.359. OrdersLiveData ... 487
14.20.360. OrdersMktAsLmtPct ... 487
14.20.361. OrdersMode ... 488
14.20.362. OrdersNetLiq ... 488
14.20.363. OrdersTemplate ... 489
14.20.364. OrdersUseQtyIn ... 490
14.20.365. OrdersUseTLEndDate ... 490
14.20.366. OrderSum ... 491
14.20.367. PadAlignSym ... 491
14.20.368. Padding ... 492
14.20.369. Parameters ... 492
14.20.370. PDI ... 493
14.20.371. Peak ... 493
14.20.372. PeakBars ... 494
14.20.373. PercentRank ... 495
14.20.374. PercentRankN ... 495
14.20.375. PointValue ... 496
14.20.376. PrevExitLimit ... 496
14.20.377. PrevExitStop ... 497
14.20.378. Product ... 497
14.20.379. PositionSum ... 498
14.20.380. PriceRound ... 498
14.20.381. QtyFinal ... 499
14.20.382. QtyPrice ... 500
14.20.383. QtyRound ... 501
14.20.384. QtyType ... 501
14.20.385. Quantity ... 502
14.20.386. Random ... 502
14.20.387. RandomSeed ... 503
14.20.388. Range or R ... 504
14.20.389. Rank ... 504
14.20.390. RankN ... 504
14.20.391. Reason ... 505
14.20.392. Reduce ... 505
14.20.393. RemoveLatest ... 506
14.20.394. RepairData ... 506

14.20.395. Replace 507
14.20.396. ReportContent 508
14.20.397. ReportsFolder 508
14.20.398. Rescale 508
14.20.399. Results 509
14.20.400. ResultsByYear 509
14.20.401. ResultsFile 510
14.20.402. Right 510
14.20.403. RiskFreeRateSym 511
14.20.404. ROC or PctChg 511
14.20.405. RollCost 511
14.20.406. Round 512
14.20.407. RSI 513
14.20.408. RRSI 513
14.20.409. RsiF 514
14.20.410. S.Alloc 514
14.20.411. S.BPx 515
14.20.412. S.CashInOut 516
14.20.413. S.Collateral / S.FuturesMargin 516
14.20.414. S.Comms 516
14.20.415. S.Compounded 516
14.20.416. S.Date 517
14.20.417. S.DDBars 517
14.20.418. S.DDDlr 517
14.20.419. S.DDPct 517
14.20.420. S.Dividends 518
14.20.421. S.Entries 518
14.20.422. S.EntryOrders 518
14.20.423. S.Equity 519
14.20.424. S.Exits 519
14.20.425. S.Exposure 519
14.20.426. S.FeesInOut 520
14.20.427. S.First 520
14.20.428. S.FreeCash 520
14.20.429. S.Interest 521
14.20.430. S.Invested 521
14.20.431. S.LongExp 521
14.20.432. S.LongInv 522
14.20.433. S.LongM2M 522
14.20.434. S.LossAmt 522
14.20.435. S.LossBars 523
14.20.436. S.Losses 523
14.20.437. S.LossPct 523
14.20.438. S.LossPctAlloc 523
14.20.439. S.M2M 523
14.20.440. S.MAE 524
14.20.441. S.MFE 524
14.20.442. S.MarkedToMarket 524
14.20.443. S.MaxAlloc 525
14.20.444. S.MaxDDBars 525
14.20.445. S.MaxDDDlr 525
14.20.446. S.MaxDDPct 525
14.20.447. S.MaxEquity 526
14.20.448. S.MinAlloc 526
14.20.449. S.MinEquity 526
14.20.450. S.NetDlr 526
14.20.451. S.NetDlrMax 527
14.20.452. S.NetDlrMin 527
14.20.453. S.NetFx 527

14.20.454. S.NetPct 528
14.20.455. S.NetPctMax 528
14.20.456. S.NetPctMin 528
14.20.457. S.Number 529
14.20.458. S.Positions 529
14.20.459. S.RiskFreeRate 529
14.20.460. S.RollCosts 530
14.20.461. S.Setups 530
14.20.462. S.ShortExp 530
14.20.463. S.ShortInv 530
14.20.464. S.ShortM2M 531
14.20.465. S.Slips 531
14.20.466. S.StartEquity 531
14.20.467. S.Stops 532
14.20.468. S.Targets 532
14.20.469. S.TradeBars 532
14.20.470. S.TradeDlr 532
14.20.471. S.TradePct 532
14.20.472. S.TradePctAlloc 533
14.20.473. S.TWEQ 533
14.20.474. S.Usage 533
14.20.475. S.WinAmt 534
14.20.476. S.WinBars 534
14.20.477. S.WinPct 534
14.20.478. S.WinPctAlloc 534
14.20.479. S.Wins 534
14.20.480. SAR 535
14.20.481. SarF 535
14.20.482. SaveAs 536
14.20.483. SaveLogAs 536
14.20.484. SavePositionsAs 537
14.20.485. SaveScanAs 537
14.20.486. SaveScanChartsTo / SaveTradeChartsTo 537
14.20.487. SaveStatsAs 539
14.20.488. SaveStatsInclude 539
14.20.489. SaveTestListAs 539
14.20.490. SaveTradesAs 540
14.20.491. SaveTradesType 540
14.20.492. Scan 541
14.20.493. ScanInclude 541
14.20.494. ScanNoDefCols 541
14.20.495. ScanNoHeader 541
14.20.496. ScanNoWindow 542
14.20.497. ScanSettings 542
14.20.498. Select 542
14.20.499. Sequence 543
14.20.500. Settings 544
14.20.501. SetupRank 544
14.20.502. SetupScore 544
14.20.503. SetupSkip 545
14.20.504. SetupSum 545
14.20.505. Shares or Contracts 546
14.20.506. ShortLoanIntPct 546
14.20.507. ShortPrcdsIntPct 546
14.20.508. Side (position) 547
14.20.509. Side (strategy) 547
14.20.510. Sign 547
14.20.511. SigSpike 548
14.20.512. SinceHigh 548

14.20.513. SinceLow 549
14.20.514. SinceTrue 549
14.20.515. Sine 550
14.20.516. Skewness 550
14.20.517. SkipNans 551
14.20.518. SkipTestIf 551
14.20.519. Slippage 552
14.20.520. Slope 553
14.20.521. Spearman 553
14.20.522. Split 554
14.20.523. Sqr 554
14.20.524. StartDate (Import) 555
14.20.525. StartDate (Setting) 555
14.20.526. StartPercent 555
14.20.527. StatsGroup 556
14.20.528. StdDev 556
14.20.529. StdErr 557
14.20.530. STOC 558
14.20.531. StopSlip 559
14.20.532. StratData 559
14.20.533. Strategy 559
14.20.534. StrategyScore 559
14.20.535. StratNum 560
14.20.536. StratRef 561
14.20.537. Sum 561
14.20.538. SumSince 561
14.20.539. SumSQ 562
14.20.540. SuperTrend 563
14.20.541. Switch 563
14.20.542. Symbol 564
14.20.543. SymChangeList 564
14.20.544. SymInfoFile 566
14.20.545. SymRef 566
14.20.546. T.Bars 566
14.20.547. T.CommIn 567
14.20.548. T.CommOut 567
14.20.549. T.DateIn 567
14.20.550. T.DateOut 567
14.20.551. T.Div 567
14.20.552. T.Fraction 568
14.20.553. T.FxIn 568
14.20.554. T.FxOut 568
14.20.555. T.Highest 568
14.20.556. T.Lowest 569
14.20.557. T.NetFx 569
14.20.558. T.NetPct 569
14.20.559. T.Points 569
14.20.560. T.PriceIn 570
14.20.561. T.PriceInZ 570
14.20.562. T.PriceOut 570
14.20.563. T.PriceOutZ 570
14.20.564. T.Profit 571
14.20.565. T.PtVal 571
14.20.566. T.QtyIn 571
14.20.567. T.QtyOut 571
14.20.568. T.Reason 571
14.20.569. T.RollCosts 573
14.20.570. T.Side 573
14.20.571. T.SlipIn 573

14.20.572. T.SlipOut 573
14.20.573. T.SplitIn 574
14.20.574. T.SplitOut 574
14.20.575. T.Strat 574
14.20.576. T.Symbol 574
14.20.577. T.TimeIn 574
14.20.578. T.TimeOut 575
14.20.579. T.ValueIn 575
14.20.580. T.ValueOut 575
14.20.581. Tangent 575
14.20.582. TanH 576
14.20.583. TargetPrice 576
14.20.584. TEMA 577
14.20.585. Template 578
14.20.586. TestInclude 578
14.20.587. TestData 578
14.20.588. TestName 579
14.20.589. TestOutput 579
14.20.590. TestScan 579
14.20.591. TestScanAllDates 580
14.20.592. TestScanPositions 580
14.20.593. TestSettings 580
14.20.594. This 580
14.20.595. TickSize 581
14.20.596. TLAdjusted 582
14.20.597. TLAdoptPos 582
14.20.598. TLCalcZPrices 582
14.20.599. TLDateFmt 583
14.20.600. TLDelim 583
14.20.601. TLEndDate 584
14.20.602. TLFields 584
14.20.603. TLNumFmt 586
14.20.604. TLStratName 586
14.20.605. TLTimeShift 586
14.20.606. TLValueIn 587
14.20.607. TLValueOut 587
14.20.608. ToDate 587
14.20.609. ToLower 588
14.20.610. ToNum 588
14.20.611. ToUpper 589
14.20.612. Top 589
14.20.613. Tracker 590
14.20.614. TradeList 590
14.20.615. Trades 591
14.20.616. TradeStatAvg 591
14.20.617. TradeStatKurt 591
14.20.618. TradeStatMax 592
14.20.619. TradeStatMin 592
14.20.620. TradeStatSkew 593
14.20.621. TradeStatStdDev 593
14.20.622. TradeStatSum 594
14.20.623. Trough 594
14.20.624. TroughBars 595
14.20.625. TrueInRow 595
14.20.626. TrueRange or TR 596
14.20.627. UntilEvent 596
14.20.628. UntilTrue 597
14.20.629. Update 597
14.20.630. UseAvailableBars 598

14.20.631. Using 598
14.20.632. Volume or V 599
14.20.633. VWMA 599
14.20.634. WalkForward 600
14.20.635. Week 600
14.20.636. WhenTrue 600
14.20.637. WMA or WAvg 601
14.20.638. Year 601
14.20.639. YInt 602

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**.

RealTest also works well alongside an AI coding assistant. See **Using RealTest with Agentic AI** for an AI-assisted approach to researching, writing, and refining trading strategies.

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**.

Thank you!

2. Getting Started

The topics in this section cover what you need to know when getting started with RealTest: the system it runs on, the file types it uses, where those files are kept, and how to back up your work.

- **Computer Requirements** - supported systems and performance notes
- **File Types** - the kinds of files RealTest reads and writes
- **File Paths** - where those files are located
- **Backups** - keeping your scripts and data safe

2.1. 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 early-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 itself is single-threaded but very efficient, although very large optimizations can be split into parallel shard processes (see **OptShards**). 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, provided that the VM emulates an x64 CPU and provides a full installation of Windows. Running RealTest on API-level emulators such as Crossover or Wine is NOT supported.

2.2. 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)
- **CSI Futures** (a highly-recommended data source for end-of-day futures)
- **CSI Stocks** (also a good source for stock data)
- **EODHD** (a good cloud-based data source with support for many international markets -- requires API key)
- **Tiingo** (another good cloud-based data source -- also requires API key)
- **Yahoo Finance** (quick and easy free data source, but with some quality issues and limitations)
- local **CSV** files (comma-delimited text -- if you have CSV data you can use it)
- local **MetaStock** databases (legacy format only)

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.

2.3. 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**
- **Scripts\Research** - a ready-to-use workspace for AI-assisted strategy research with **Claude Code**
- **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.

2.4. Backups

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

Whenever a script file (*.rts) is saved, the prior version is first copied to the backups folder and renamed to add a version number, e.g. myscript(1).rts.

Version numbers of previously backed up scripts are first incremented, so that (1) is always the most recent backup, (2) the second most recent, etc.

The maximum number of backups of each file to keep is controlled by a setting in the **Script Editor Options** dialog box and defaults to 3.

The RealTest.ini configuration file is also automatically backed up each time it is saved, using the same mechanism and max backups setting.

No other file types are backed up as the others (data, results, logs, etc.) can be recreated by re-running scripts.

It is, of course, highly recommended to add your RealTest scripts folder(s) and the backup folder to your list of folders to be backed up by whatever automatic local or cloud backup services you are using, i.e., to also maintain backups of these critical files in a location other than your working computer.

When configuring external backups of RealTest files and folders, it is advisable to exclude data files (**.rtd*), 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 your anti-virus scanning software for better performance.

3. Brief Tutorials

These topics introduce RealTest through four short, guided tutorials. There is also a series of tutorial videos on the **mhptrading YouTube channel**.

- **Tutorial 1 - SPY Crossover** - a moving-average crossover system for SPY
- **Tutorial 2 - Simple Optimization** - parameterizing and optimizing a strategy
- **Tutorial 3 - Simple Scan** - the Data section and the scanner
- **Tutorial 4 - ETF Rotation** - a monthly sector-ETF rotation strategy

3.1. 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).

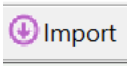
When you first run RealTest, you will see that this example script has already been opened for you in a **Script Window**:

```
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
            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

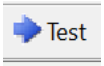
To start, click on "Import" in the Tool Bar at the top of the main window.



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:

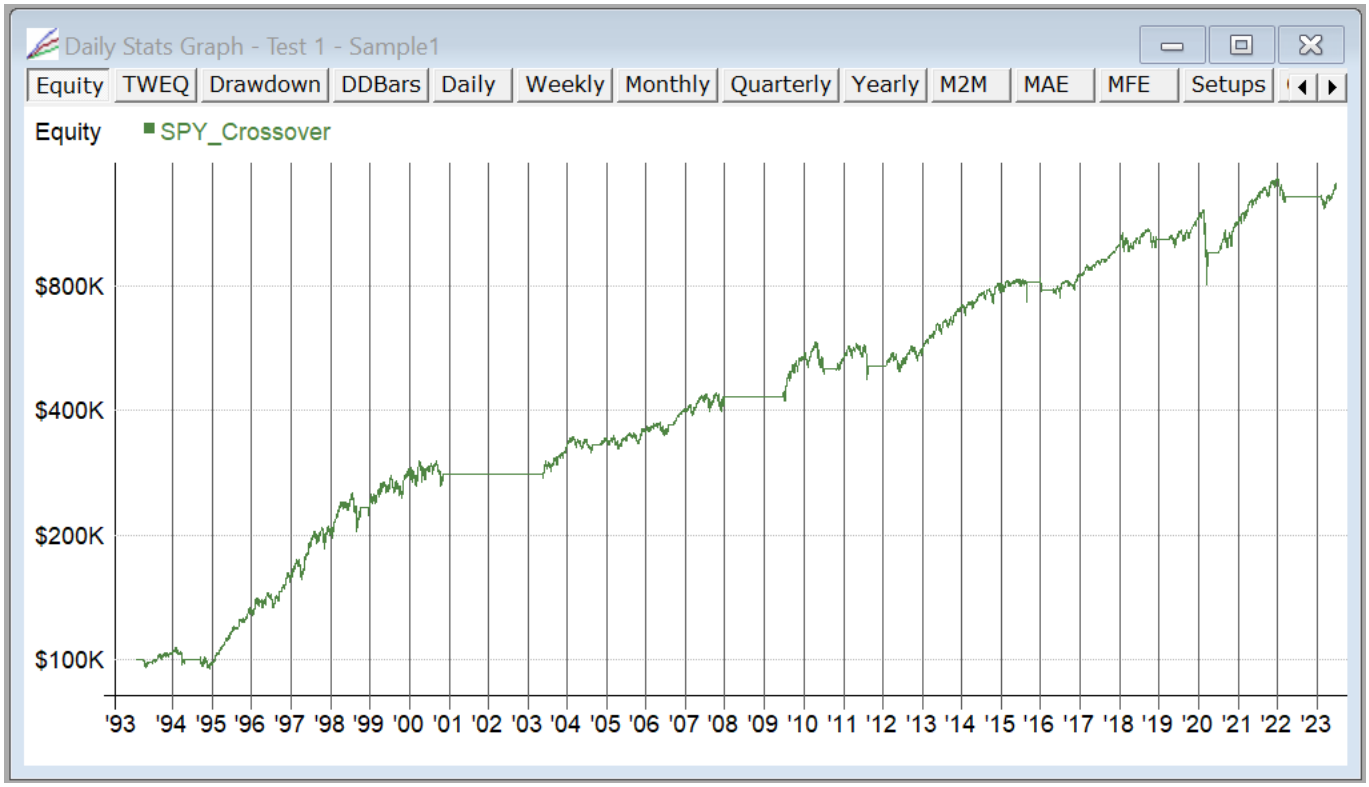
Results - (untitled)

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%

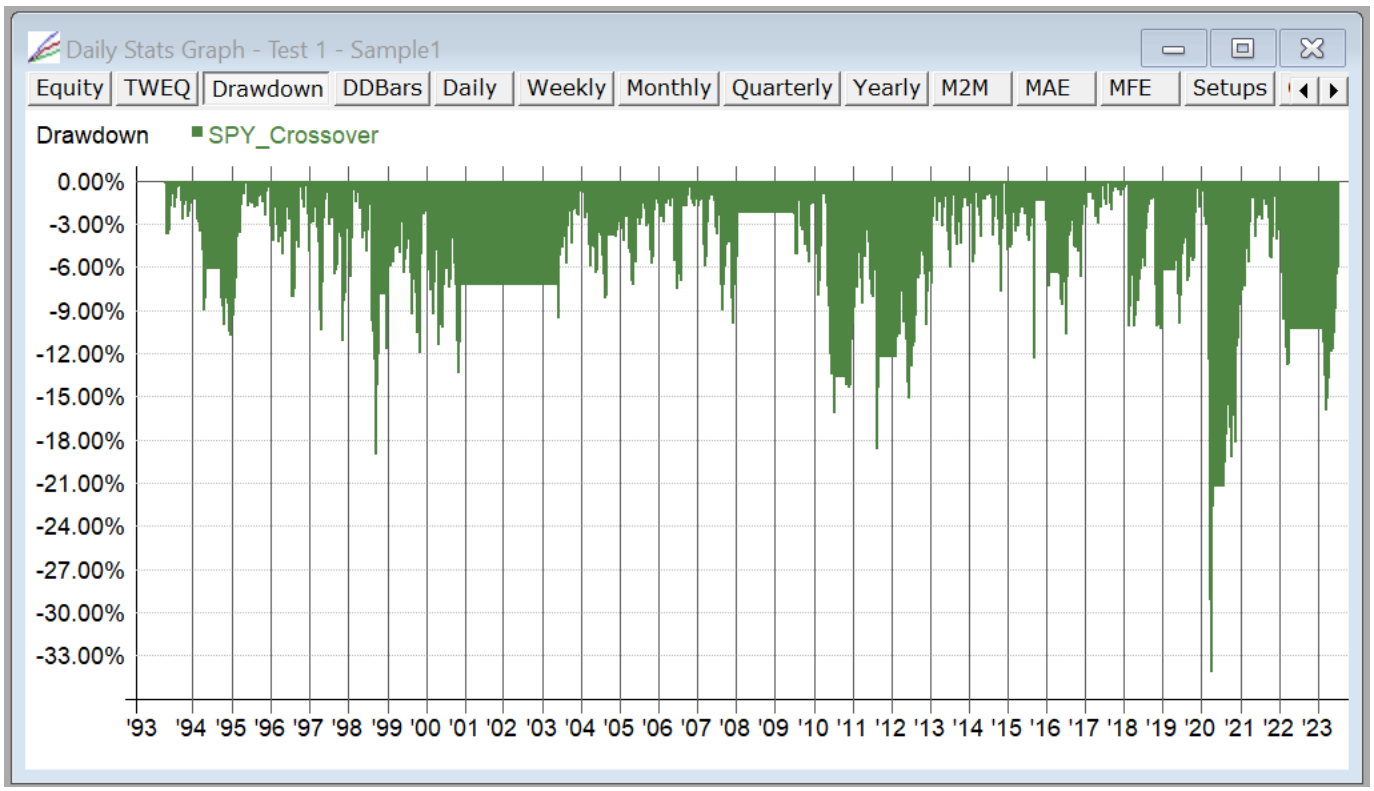
(There are other default columns, but they'd make the image too wide to be readable here.)

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

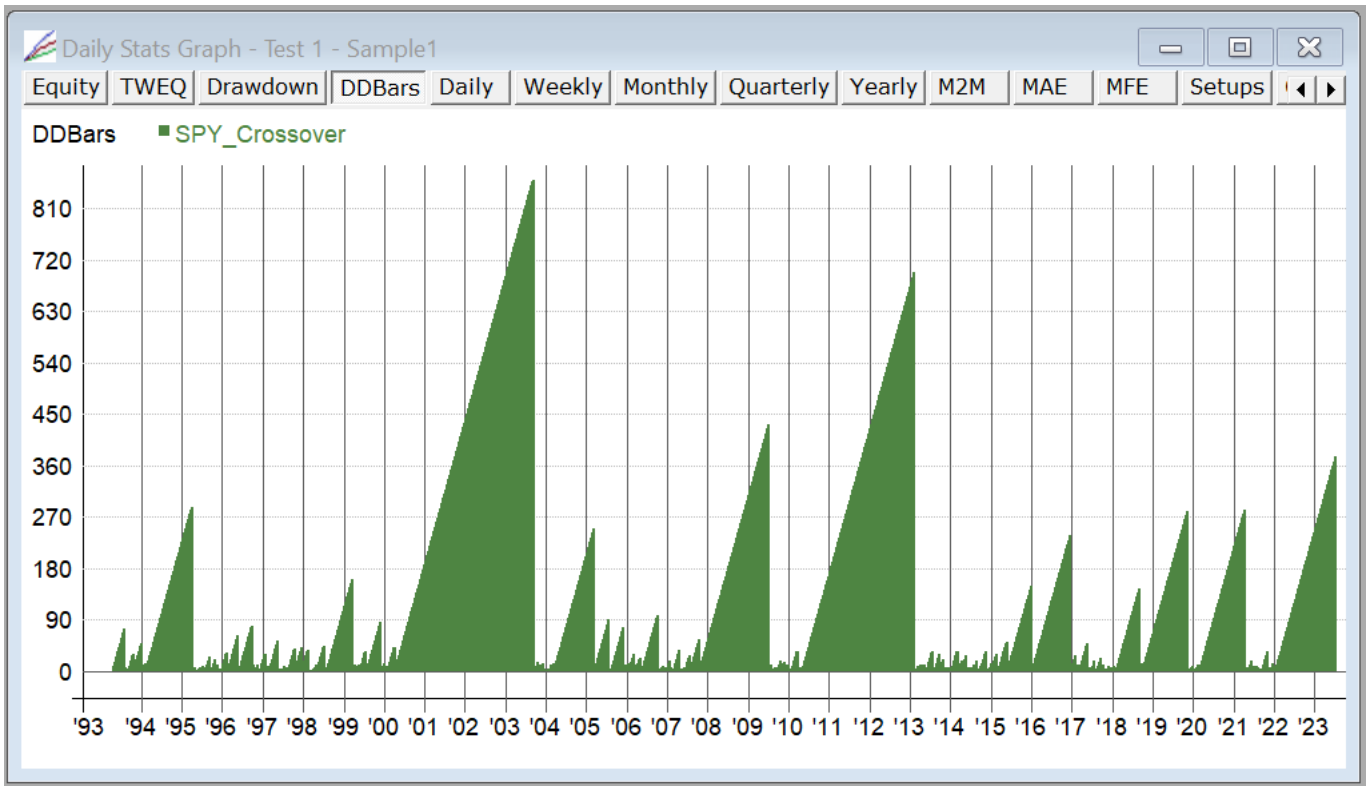
The **Daily Stats Graph** shows the equity curve:



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.

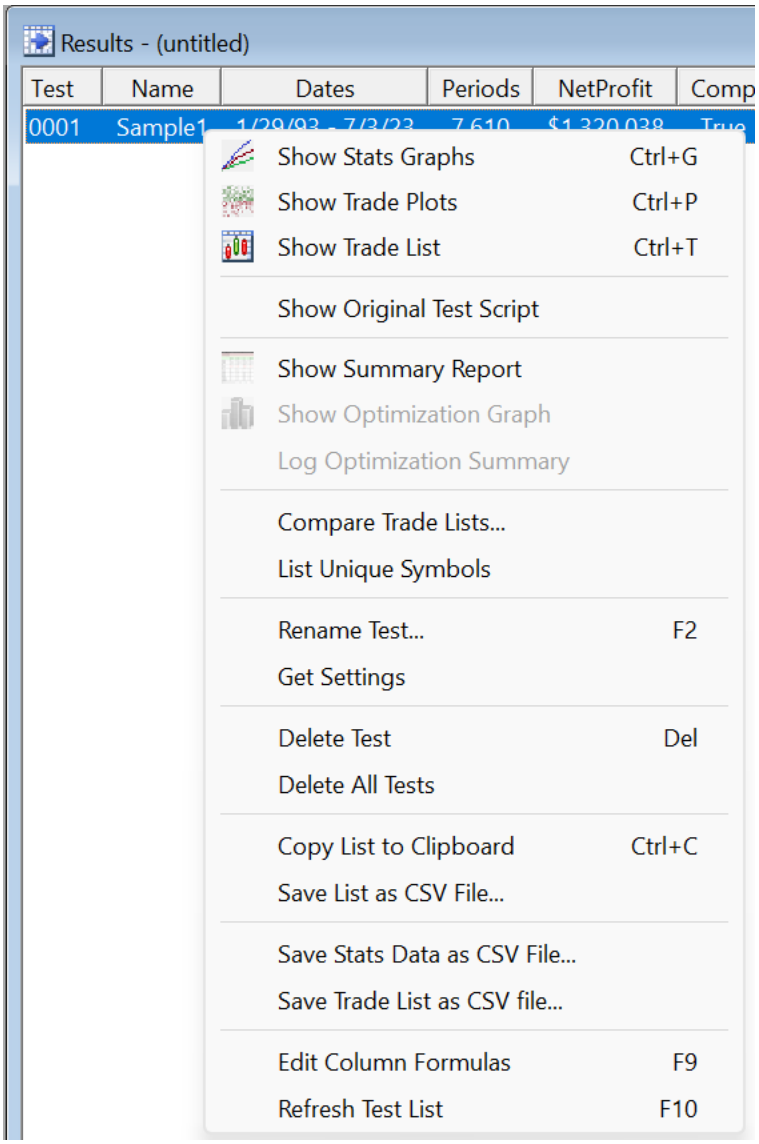


You can also use the left and right arrow keys to cycle through the different graph types.

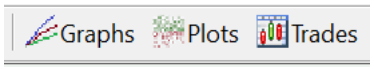
When there is more than one test in the results window, the up and down arrow keys let you easily see the same graph for different tests.

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:



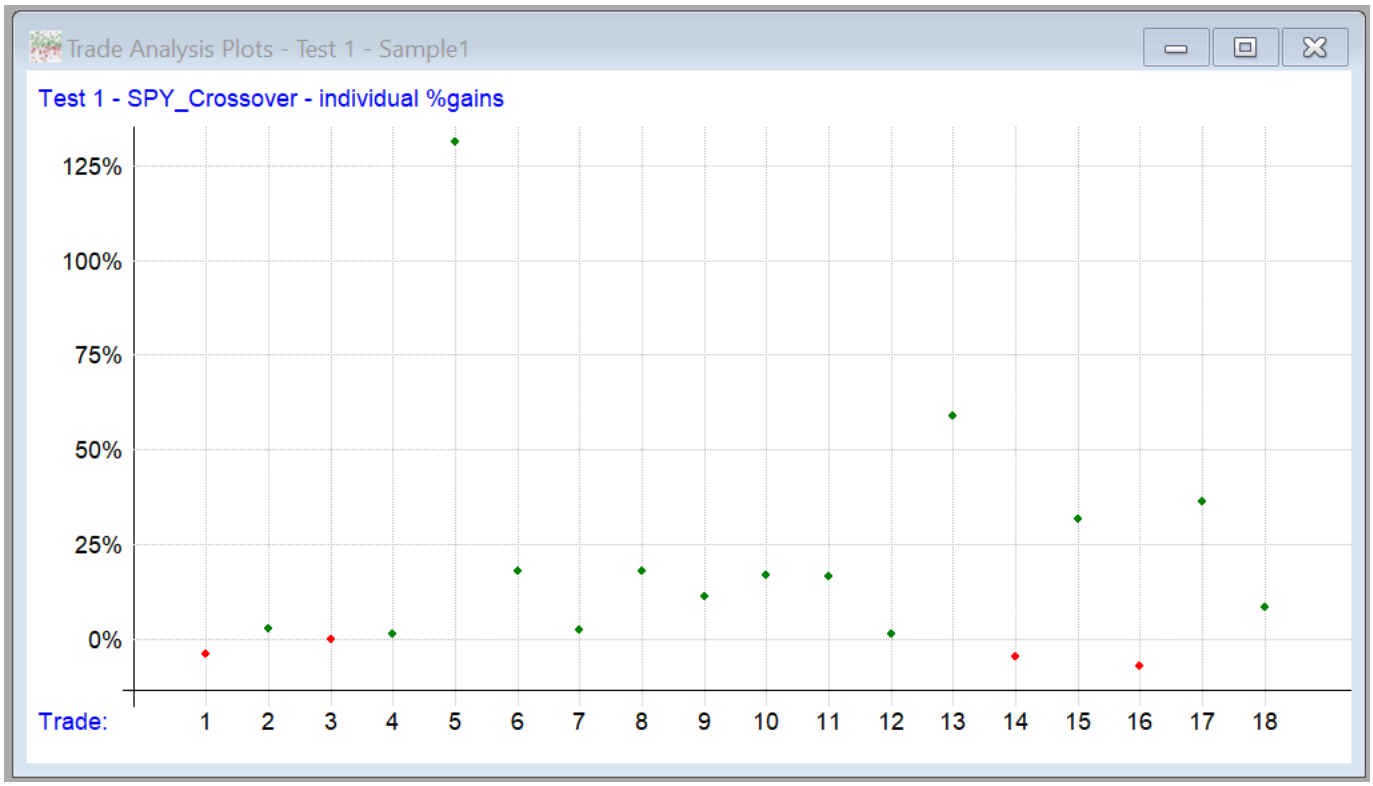
The first three items are the ones you'll use most often, so they also appear in the Tool Bar:



(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 List - Test 1 - 18 Trades

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.

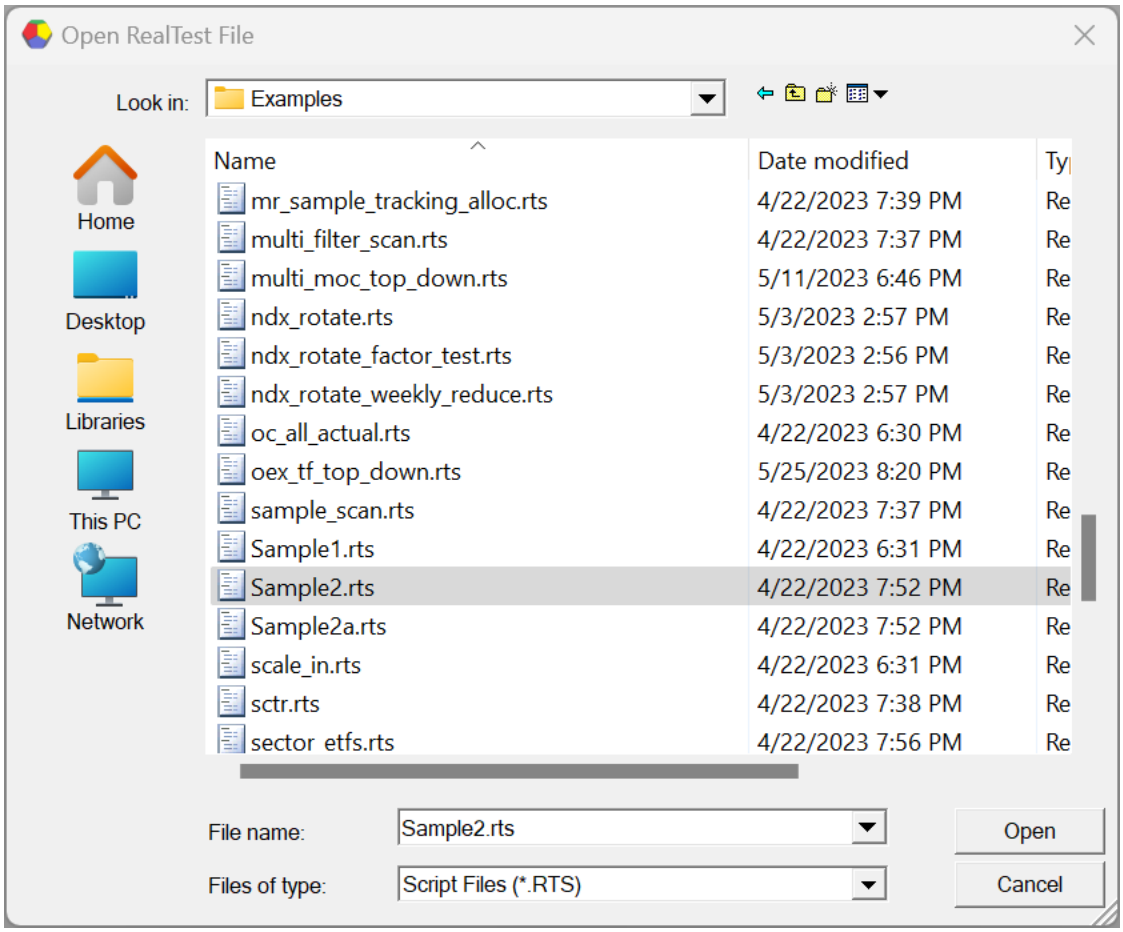
That's the end of this first tutorial!

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

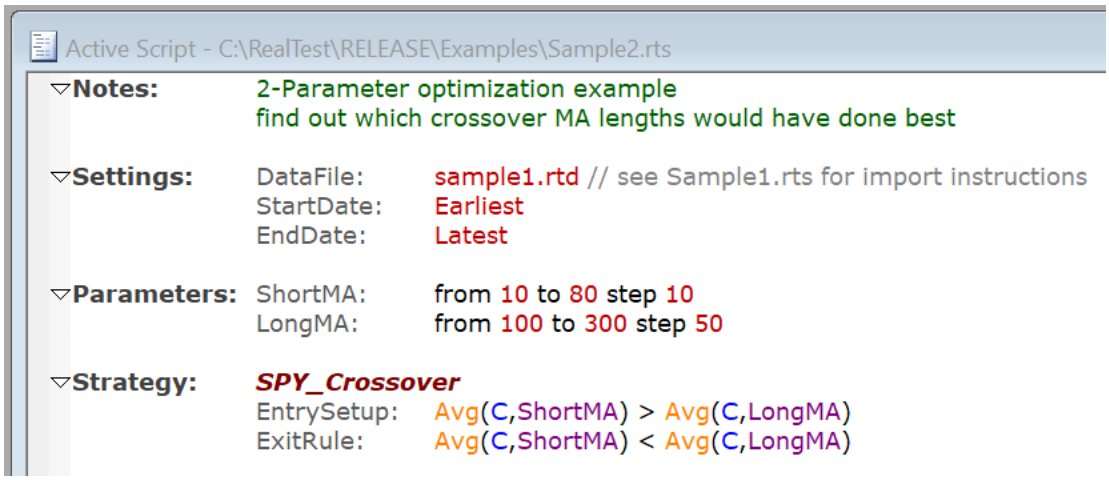
3.2. Tutorial 2 - Simple Optimization

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.

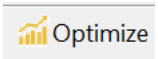


This script will use the same data file as the one in the first tutorial, so there is no Import section.

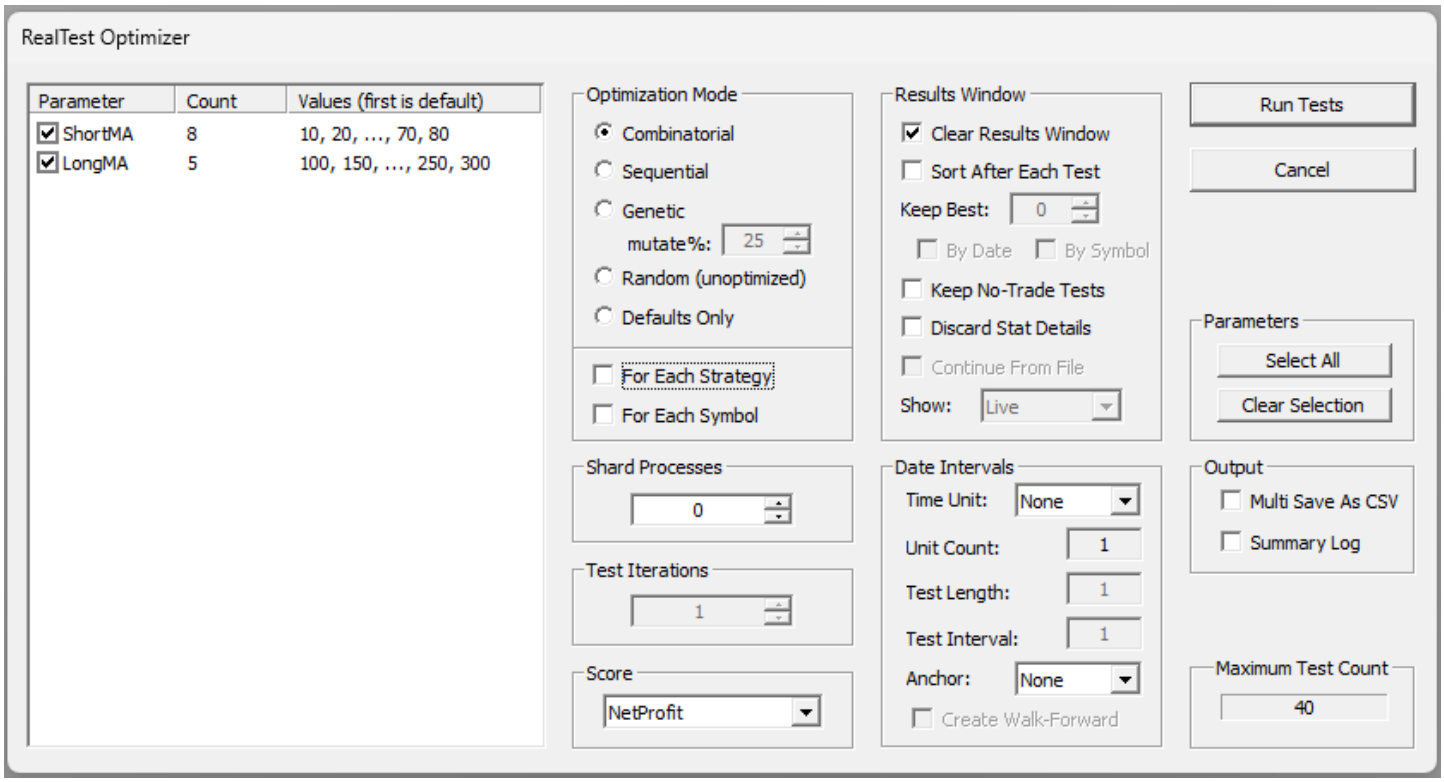
Rather than running a single test, we will now try the **RealTest Optimizer**.

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.

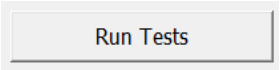
Press 

and the following dialog will appear:



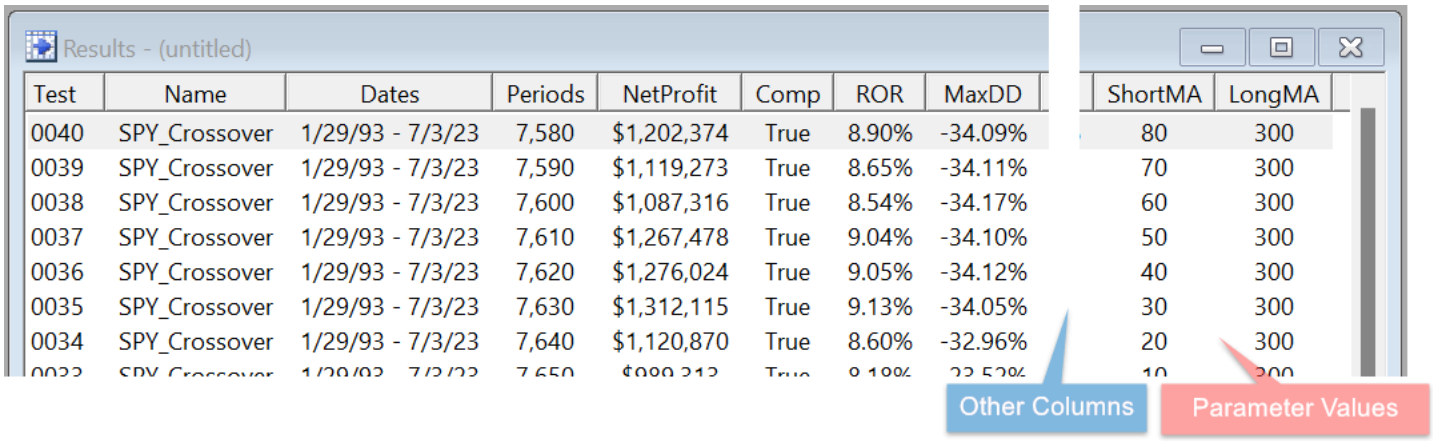
For now, you can ignore most of the settings in this dialog. Just focus on the upper-left corner, where the **Parameters** that were defined in the script are shown. Also notice the lower-right corner, where the number of tests to be run is calculated and displayed.

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.



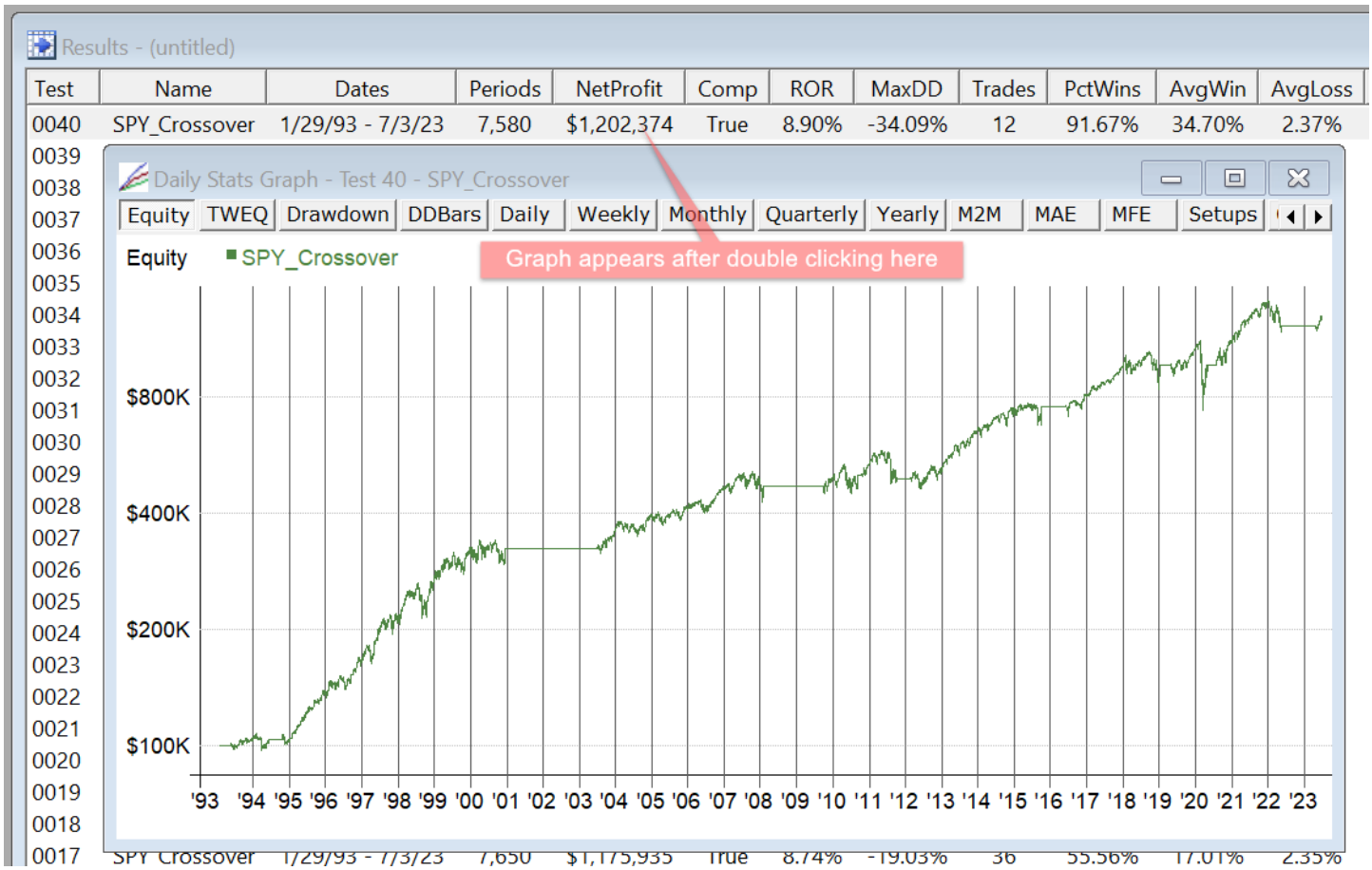
Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	ShortMA	LongMA
0040	SPY_Crossover	1/29/93 - 7/3/23	7,580	$1,202,374	True	8.90%	-34.09%	80	300
0039	SPY_Crossover	1/29/93 - 7/3/23	7,590	$1,119,273	True	8.65%	-34.11%	70	300
0038	SPY_Crossover	1/29/93 - 7/3/23	7,600	$1,087,316	True	8.54%	-34.17%	60	300
0037	SPY_Crossover	1/29/93 - 7/3/23	7,610	$1,267,478	True	9.04%	-34.10%	50	300
0036	SPY_Crossover	1/29/93 - 7/3/23	7,620	$1,276,024	True	9.05%	-34.12%	40	300
0035	SPY_Crossover	1/29/93 - 7/3/23	7,630	$1,312,115	True	9.13%	-34.05%	30	300
0034	SPY_Crossover	1/29/93 - 7/3/23	7,640	$1,120,870	True	8.60%	-32.96%	20	300

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.

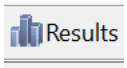
Click the same column again to reverse the sort order. Shift-click other columns to create a multi-level sort.

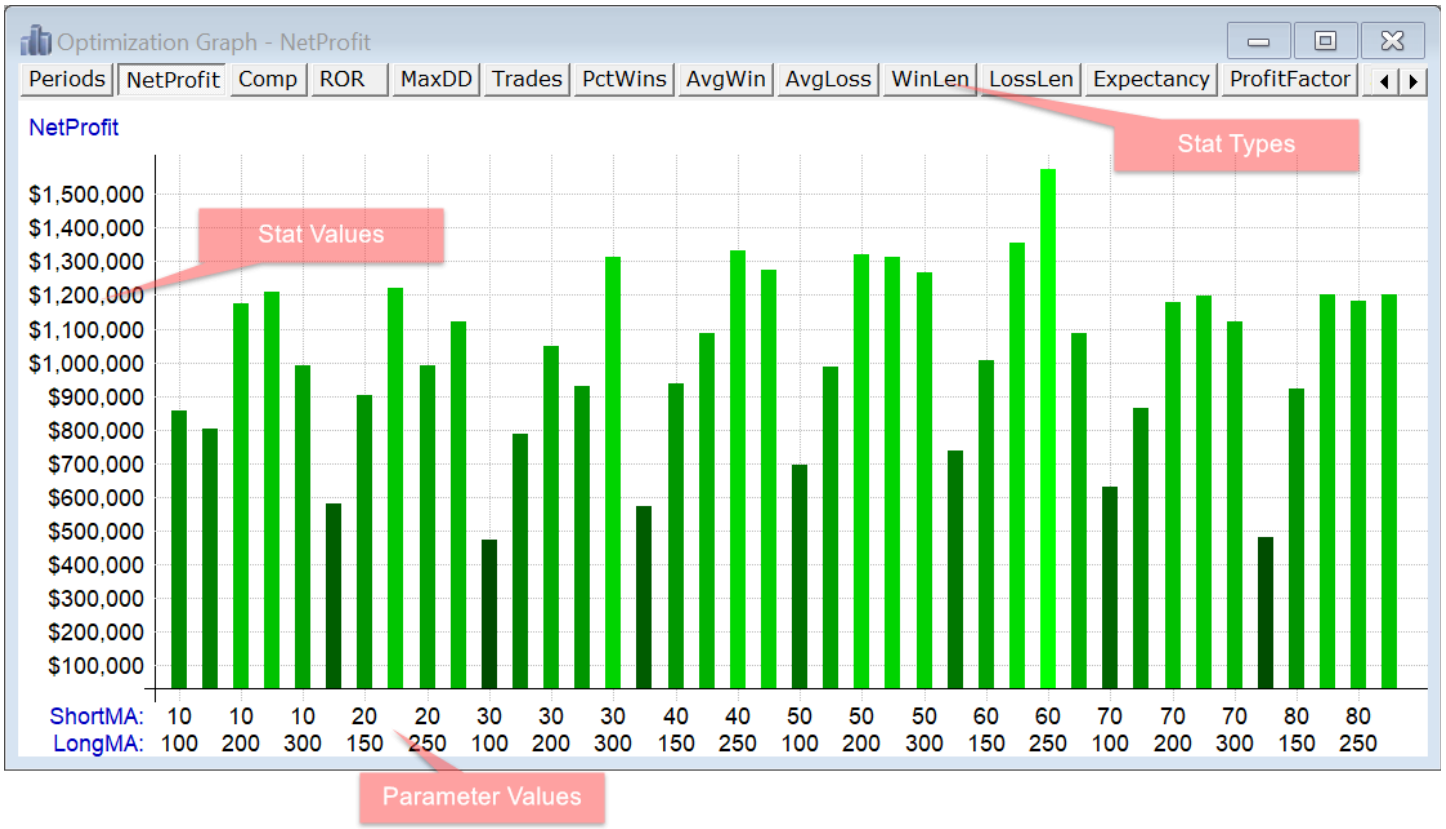
After sorting the results list by ascending NetProfit, double-click on the top row to open the Daily Stats Graph showing the equity curve.



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).

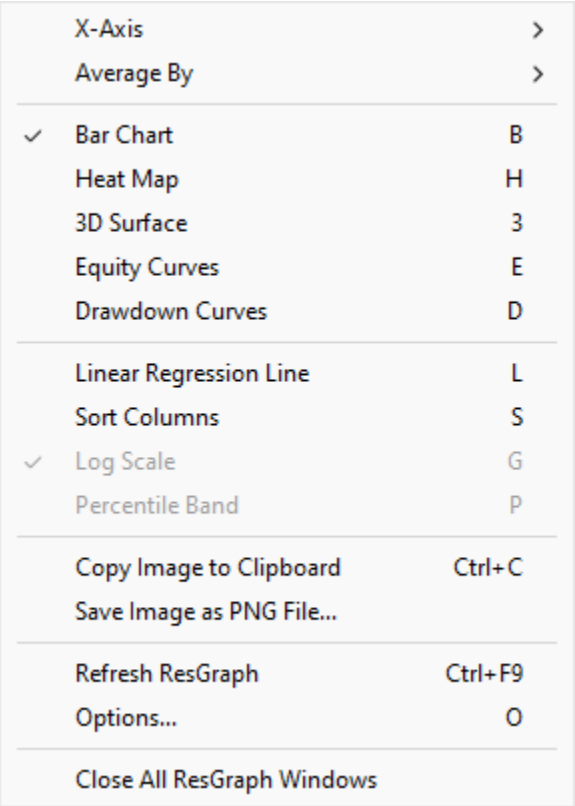
To get a better sense of the relationship between these parameters and the corresponding test results, click on  in the Tool Bar to open the **Optimization Results Graph**.



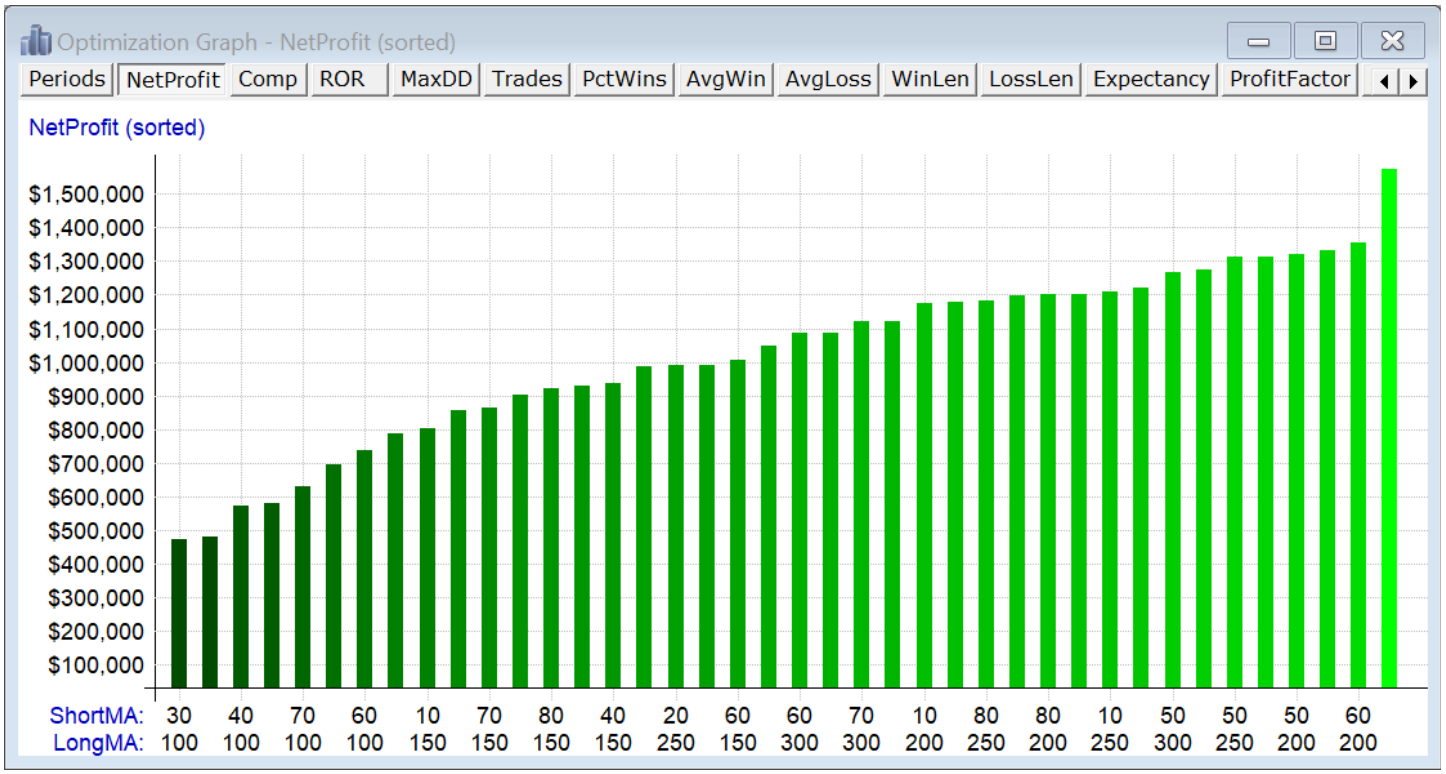
If you see a graph other than NetProfit, use the button bar to select NetProfit. Every column from the Results window can be graphed here.

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.



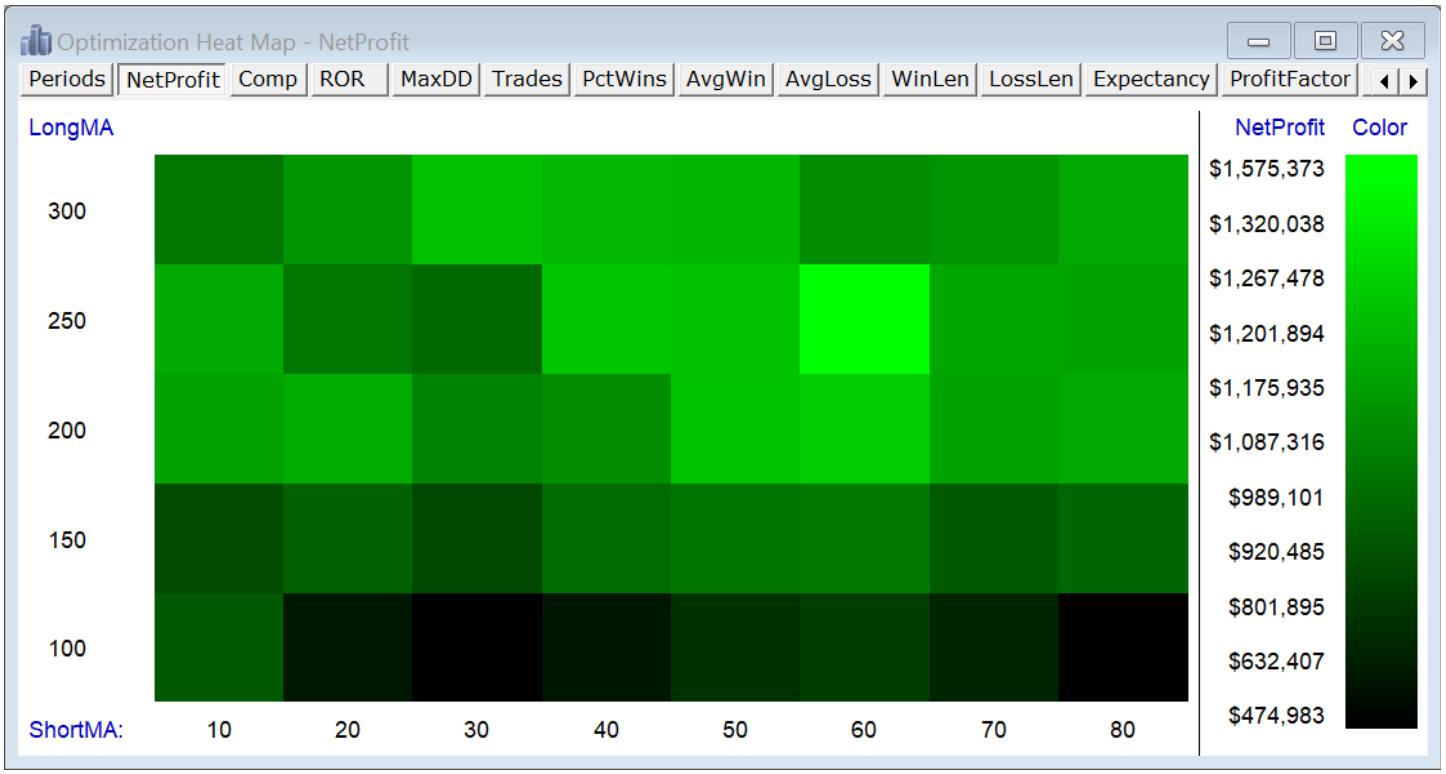
Selecting *Sort Columns* changes the graph to this:



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**.

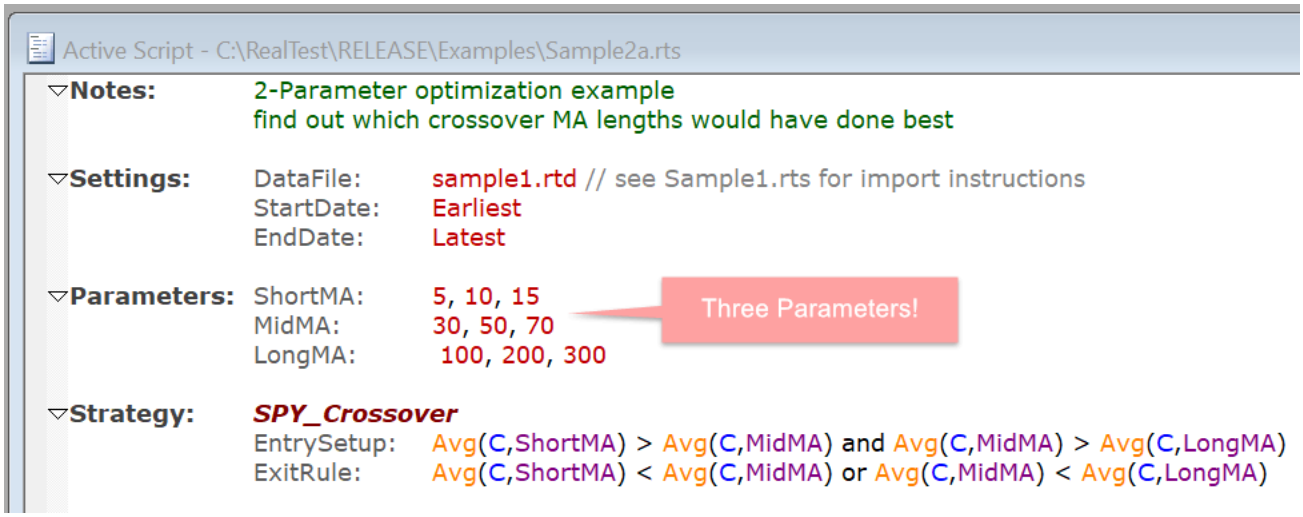
Now you'll see this nice-looking checkerboard:



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).

(The same two-parameter results can also be shown as a rotating 3D **Surface** - press *3* or pick it from the menu - which tilts the heat-map grid into three dimensions. It carries the same information as the flat heat map; many people find the flat view quicker to read, but the raised surface can make the overall shape easier to take in. The menu offers several more views as well, including equity and drawdown curves for every test at once; see **Optimization Results Graphs** for the complete set.)

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



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

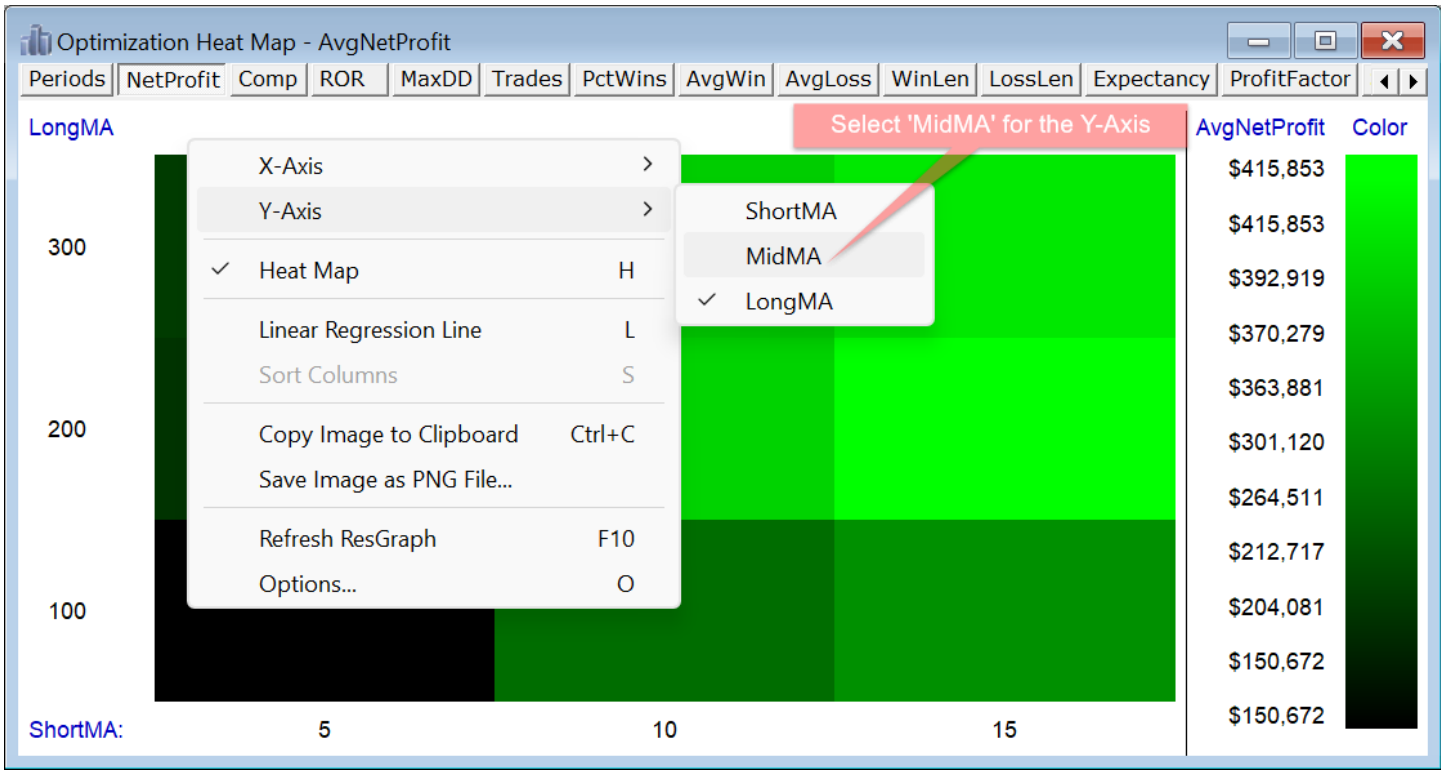
It will open as a **Heat Map**, since that's the previous view it was shown in.

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 **Multiple Tests and Optimization** topic.

3.3. 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).

```
Active Script - C:\RealTest\RELEASE\Examples\import_yahoo.rts*
Notes:   Import data for the current S&P 500 components (plus SPY) from Yahoo

Import:  DataSource:   Yahoo
         IncludeList:  ?scriptpath?\spx_syms.txt
         IncludeList:  SPY
         StartDate:    1/1/15
         SaveAs:       YahooSPX.rtd
```

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.**

```
Active Script - C:\RealTest\SCRIPTS\Examples\sample_scan.rts
Notes:    Scans for SPX components down 25% within 63 days

Settings: DataFile:   YahooSPX.rtd // see import_yahoo.rts for import definition
          EndDate:    Latest
          NumDays:    1

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
          Sort:       Drop
```

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:

Scan - 38 Items

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.

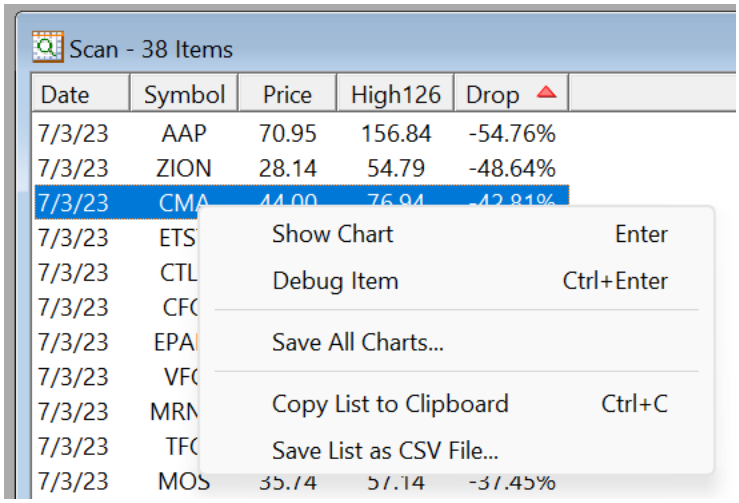
At the time of this writing, 38 stocks met this condition.

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:



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.

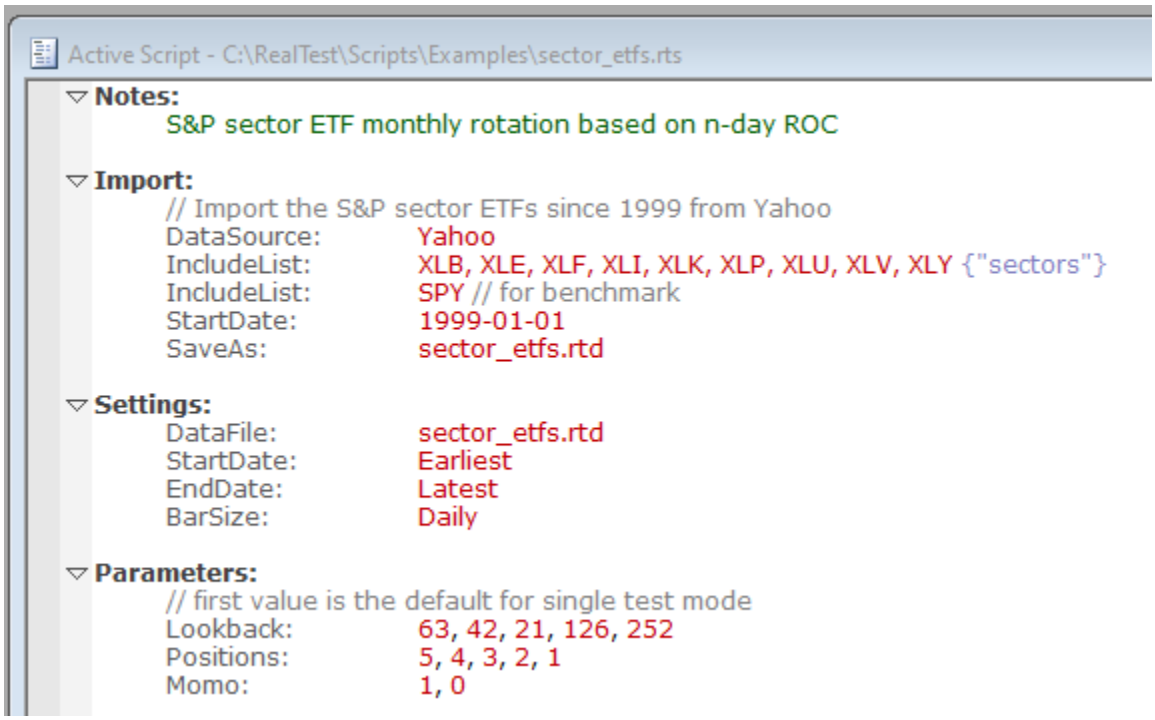
There is also a **SaveScanAs** option in the **Settings** section, which, if specified, causes the scan output to be automatically saved every time the scan is run. This can be useful for something like producing a daily trading candidate list.

That's the end of this third tutorial!

3.4. 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.



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
    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
    MaxPositions:   Positions
    Commission:     0.005 * Shares
    Slippage:       0.001 * FillPrice
    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

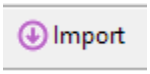
data formulas to enable some optimization.

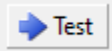
```
Parameters:
    // first value is the default for single test mode
    Lookback:       63, 42, 21, 126, 252
    Positions:      5, 4, 3, 2, 1
    Momo:           1, 0

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):

Results - (untitled)

Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	MAR	Trades	PctWins	AvgWin	AvgLoss
0001	sector_etfs	1/4/99 - 6/30/26	6,832	$517,670	True	6.87%	-58.40%	0.12	470	52.34%	9.57%	5.55%

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)
☑ Lookback	5	63, 42, ..., 126, 252
☐ Positions	5	5, 4, ..., 2, 1
☐ Momo	2	1, 0

Run the 5 tests and look at the results:

Results - (untitled)

Test	Name	Dates	Periods	NetProfit	Comp	ROR	MaxDD	Lookback	Positions	Momo
0005	sector_etfs	1/4/99 - 6/30/26	6,662	$711,960	True	8.32%	-59.99%	252	5	1
0004	sector_etfs	1/4/99 - 6/30/26	6,769	$551,384	True	7.21%	-60.76%	126	5	1
0003	sector_etfs	1/4/99 - 6/30/26	6,876	$535,960	True	7.00%	-57.84%	21	5	1
0002	sector_etfs	1/4/99 - 6/30/26	6,853	$422,040	True	6.27%	-58.05%	42	5	1
0001	sector_etfs	1/4/99 - 6/30/26	6,832	$517,670	True	6.87%	-58.40%	63	5	1

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	5	0

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

case.

In fact, most of the best results are using this anti-momentum factor:

Lookback	Positions	Momo
42	5	0
126	2	0
42	4	0
126	3	0
252	1	1
42	3	0
126	1	0
126	4	0
126	5	0
21	5	0
42	1	0
63	3	0
252	2	1
63	5	0
63	4	0
252	5	1
21	4	0
252	5	0
21	3	0
63	1	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**.

4. Classic Example Scripts

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_eodhd.rts** - shows how to import data from EODHD, if you have an API key
- **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
- **correl_multi.rts** - demonstrates the multi-symbol correlation functions by building a correlation matrix for DJIA constituents
- **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
- **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
- **supertrend.rts** - shows how to calculate and plot the "supertrend" indicator (dual trailing stops)
- **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** - implements this strategy from a well-known book (dynamic sizing is done the old way)
- **clenow_stocks_on_move_dynamic.rts** - implements the same strategy as above using dynamic sizing mode
- **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
- **hybrid_asset_allocation_dynamic.rts** - implements the above with partial entries and exits
- **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_make_tradelist.rts** - makes trade list for use with the *ndx_plus_wtf_tradelists.rts* example
- **ndx_rotate_rebalance.rts** - monthly rebalance version using dymanic sizing mode

- **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"
- **optimal_diversity.rts** - uses "greedy scoring" to calculate an optimally diversified portfolio from the S&P 100 universe
- **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
- **simple_day_trade_factor_scan.rts** - demonstrates how to add trade factors to a script for this analysis technique
- **simple_day_trade_symbol_track.rts** - version of simple_day_trade with per-symbol recent-performance tracking
- **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.rts** - 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_follow.rts** - a weekly Russell 1000 trend-following strategy similar to the Nick Radge "Weekend Trend Trader"
- **weekly_trend_follow_make_tradelist.rts** - makes trade list for use with the *ndx_plus_wtf_tradelists.rts* example

---

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
- **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
- **ndx_plus_wtf_script.rts** - combines two single-strategy examples into one script directly
- **ndx_plus_wtf_tradelists.rts** - combines two single-strategy examples by playing back their trade lists together
- **oex_tf_top_down.rts** - combines three OEX trend-following strategies using top-down setup selection
- **oex_tf_auto_track.rts** - demonstrates use of auto-generated tracker benchmarks to favor strategies with the best recent performance
- **two_accounts.rts** - compound two strategies together in one account with combined with a third strategy in a separate account
- **two_accounts_rebalance.rts** - same as above with addition of periodic cash transfers to rebalance the accounts

---

Mean Reversion Theme and Variations

- **mr_sample.rts** - a simple long/short mean-reversion strategy pair using Norgate data for Russell 1000 historical constituents
- **mr_sample_common.rts** - contains the common elements of the following five examples, each of which includes this script
- **mr_sample_benchmark.rts** - shows how to add an index benchmark line to the equity curve for a set of strategies
- **mr_sample_hedged.rts** - shows how to add a dynamic-sizing strategy that hedges long-short exposure differences at each day's close
- **mr_sample_orders.rts** - shows how to configure this pair of strategies for OrderClerk order generation
- **mr_sample_tracking.rts** - shows how to test a strategy that only takes trades when its own equity curve is above a moving average
- **mr_sample_tracking_alloc.rts** - uses benchmark-copy equity curves to dynamically adjust each strategy's allocation (vs. mr_sample_tracking.rts, which uses them as on/off switches)
- **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_csi.rts** - long/short trend-following example using CSI back-adjusted continuous contract data to model individual contract trading
- **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
- **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
- **exchangemap_csi.csv** - example **ExchangeMap** file with CSI futures exchange names
- **exchangemap_ndu.csv** - example *ExchangeMap* file with Norgate futures exchange names
- **holidays.au.txt** - example Australian market **HolidayList** file to use when generating orders (or simply use *HolidayList: au_auto*)
- **holidays.ca.txt** - example Canadian market *HolidayList* file to use when generating orders (or simply use *HolidayList: ca_auto*)
- **holidays.us.txt** - example US market *HolidayList* file to use when generating orders (or simply use *HolidayList: us_auto*)
- **ibfutures_csi.csv** - futures symbol mapping from CSI to IB (add to script with **SymChangeList** if needed)
- **ibfutures_ndu.csv -** futures symbol mapping from Norgate to IB (ditto)
- **ib_basket_template.csv** - order list **template file** for IB Basket Trader output format
- **spx_syminfo.csv** - a detailed symbol information metadata file for the S&P 500 components
- **spx_syms.txt** - symbols of the current S&P 500 components

5. New Example Scripts

The **Examples-New** folder is a curated library of example scripts organized by **what you're trying to build**, not by syntax feature. You'll find it in the *Scripts\Examples-New* subfolder of wherever you installed RealTest - *C:\RealTest\Scripts\Examples-New* for a default installation.

Every script is self-contained: it carries its own Import section (run Import once per data file, then Test as often as you like), states its teaching purpose in Notes, and lists the features it demonstrates. Scripts within a folder share a data file (the same SaveAs name), so importing any one of them covers its siblings. Most use Norgate data; the *10_data_import* folder covers every other source.

Folder Contents

01_basics

The on-ramp: five short scripts that introduce the core moving parts of a RealTest script. Read them in order - every later example assumes them.

- **first_backtest.rts** - your first backtest: a complete, minimal RealTest script
- **entry_types.rts** - entry order types: the same dip-buy idea traded three ways
- **exit_types.rts** - profit target, protective stop, and rule-based exits working together
- **position_sizing.rts** - the four ways to size a position, side by side
- **data_vs_library.rts** - Data vs. Library: the most important distinction in the script language

---

02_mean_reversion

Dip-buying and spike-fading on a survivorship-free stock universe.

- **mr_pullback_long.rts** - the canonical dip-buy on a liquid stock universe
- **mr_short_side.rts** - fading upward spikes
- **mr_long_short_pair.rts** - long and short mean reversion combined in one script
- **mr_pair_common.rts** - shared components included by mr_long_short_pair.rts (not meant to be run directly)
- **mr_moc_daytrade.rts** - day trading: buy an intraday flush, sell the same day's close
- **mr_scale_in.rts** - scaling in: averaging into a dip with multiple entries per symbol

---

03_trend_breakout

Breakouts, trailing stop constructions, weekly bars, and partial exits.

- **tf_donchian.rts** - trend following basics: buy breakouts, ride a trailing stop, size by risk
- **tf_trailing_stops.rts** - trailing stop techniques: seven exits for one entry, selectable by parameter
- **tf_weekly.rts** - a strategy on weekly bars, honestly measured on daily bars
- **tf_pullback_entry.rts** - buying the pullback in a trend: momentum continuation with a stop-limit entry
- **tf_scale_out.rts** - scaling out: sell half at the first target, trail the rest

---

04_rotation_allocation

Momentum rotation and ETF asset-allocation portfolios.

- **rotate_monthly.rts** - monthly momentum rotation: hold the strongest stocks in an index
- **rotate_dynamic.rts** - rotation with true rebalancing using DynamicSizing mode
- **alloc_fixed_weights.rts** - a five-asset fixed-weight "all weather" style portfolio

- **alloc_dual_momentum.rts** - monthly tactical asset allocation: dual momentum with a canary asset
- **alloc_vol_target.rts** - inverse-volatility weighting ("risk parity lite")

05_multi_strategy

The three capital-sharing patterns, cross-strategy caps, benchmarks and trackers.

- **combine_shared_capital.rts** - combining strategies, pattern 1: one shared capital pool
- **combine_start_percent.rts** - combining strategies, pattern 2: StartPercent sleeves
- **combine_two_accounts.rts** - combining strategies, pattern 3: multiple accounts with cash transfers
- **combine_strategy_score.rts** - strategy-level competition: StrategyScore and oversubscribed capacity
- **combine_benchmark.rts** - benchmarks and trackers: comparison overlays and equity-curve gating

---

06_market_context

Regime, breadth, sector, and volatility context - with honest A/B tests of each filter, including one that fails.

- **regime_filter.rts** - gate a strategy on the index trend, and test whether the gate helps
- **breadth_filter.rts** - market breadth: build internals from the universe itself
- **sector_limits.rts** - sector ranking and concentration limits
- **vix_filter.rts** - using VIX level and term structure in a strategy
- **index_membership.rts** - what survivorship-bias-free testing actually means

---

07_futures

Continuous-contract trend following, cost sensitivity, individual contracts and term structure. These scripts require custom Norgate futures watchlists - see each script's Notes.

- **futures_trend.rts** - diversified trend following on continuous contracts, long and short
- **futures_one_contract.rts** - one-contract testing and cost sensitivity
- **futures_term_structure.rts** - individual contracts and term structure: VIX futures long/short

---

08_optimization

Parameter sweeps, pruning, walk-forward, overfitting analysis, and a random-entry null distribution.

- **opt_grid.rts** - your first optimization: sweep two parameters, read the surface
- **opt_constraints.rts** - optimizer controls: pruning, skipping, and ranking a sweep
- **opt_walk_forward.rts** - walk-forward testing: simulate periodic re-optimization, honestly
- **opt_overfitting.rts** - overfitting analysis: how much of your optimization is curve-fit?
- **opt_random_baseline.rts** - random-entry baseline: the null hypothesis for your edge

---

09_scans_orders

From research to live: scans, order lists, broker baskets, rebalance files, and replaying actual fills.

- **scan_basic.rts** - a basic scan: today's signal list, not a backtest
- **scan_multi_filter.rts** - one scan, several signal lists
- **orders_tomorrow.rts** - tomorrow's orders: from backtest to live trading

- **orders_basket.rts** - broker basket orders: CSV output for IB Basket Trader
- **orders_rebalance.rts** - target-percent output for the IB rebalance tool
- **track_actual_trades.rts** - replaying actual trades: stats and equity from a list of real fills

The folder also includes the holidays.us.txt, ib_basket_template.csv and sample_trades.csv files used by these scripts.

10_data_import

One example per data source, plus symbol metadata, event lists, and external data series.

- **import_norgate.rts** - the full-service data source
- **import_yahoo.rts** - free data, no account needed
- **import_eodhd.rts** - global exchange coverage via API
- **import_tiingo.rts** - a low-cost API data source
- **import_csv.rts** - your own data, one CSV file per symbol
- **import_metastock.rts** - reading MetaStock-format local data
- **import_multi.rts** - one data file from several sources
- **syminfo_events.rts** - symbol information files and event lists: metadata and dated events
- **external_series.rts** - joining your own values to the bars

The folder also includes sample data used by these scripts: djia_earnings.csv, djia_info.csv, djia_syms.txt, sample_values.csv, and the sample_csv folder.

11_output_reporting

Custom Results, Trades, Charts and Graphs sections, TestData/StratData, and account-level cash flows and fees.

- **custom_results.rts** - design your own results table
- **custom_trades_charts.rts** - custom trade list columns and chart overlays
- **custom_graphs.rts** - daily series for the stats graph and CSV export
- **testdata_stratdata.rts** - custom analytics computed during the test
- **accounting.rts** - deposits, fees, cash interest, and honest Sharpe

---

12_special_topics

International markets, multi-currency accounts, Library functions with arguments, and a debugging checklist.

- **international_asx.rts** - trading a non-US market: Australian stocks, weekly MOC rotation
- **multi_currency.rts** - US, Australian, and Canadian stocks in one account
- **library_functions.rts** - reusable formulas with arguments
- **debug_and_validate.rts** - debugging and validating a strategy that "runs but looks wrong"

The folder also includes the holidays.au.txt file used by these scripts.

Which Example Do I Want?

I want to...	Start here
Run my first backtest	*01_basics\first_backtest.rts*
Understand entry/exit order types	*01_basics\entry_types.rts*, *exit_types.rts*
Size positions (shares / dollars / percent / risk)	*01_basics\position_sizing.rts*

Understand Data vs Library (do this early!)	*01_basics\data_vs_library.rts*
Buy dips in a stock universe	*02_mean_reversion\mr_pullback_long.rts*
Short spikes / trade both sides	*02_mean_reversion\mr_short_side.rts*, *mr_long_short_pair.rts*
Day trade (no overnight holds)	*02_mean_reversion\mr_moc_daytrade.rts*
Average into a position	*02_mean_reversion\mr_scale_in.rts*
Trade breakouts with trailing stops	*03_trend_breakout\tf_donchian.rts*, *tf_trailing_stops.rts*
Trade weekly bars	*03_trend_breakout\tf_weekly.rts*
Buy pullbacks in trends (stop-limit entries)	*03_trend_breakout\tf_pullback_entry.rts*
Take partial profits	*03_trend_breakout\tf_scale_out.rts*
Rotate into the strongest stocks monthly	*04_rotation_allocation\rotate_monthly.rts*
Rebalance positions to target sizes	*04_rotation_allocation\rotate_dynamic.rts*
Run a fixed-weight ETF portfolio	*04_rotation_allocation\alloc_fixed_weights.rts*
Tactical asset allocation (dual momentum)	*04_rotation_allocation\alloc_dual_momentum.rts*
Risk-parity style weighting	*04_rotation_allocation\alloc_vol_target.rts*
Combine several strategies in one account	*05_multi_strategy\combine_shared_capital.rts* (start), then *combine_start_percent.rts*, *combine_two_accounts.rts*
Let strategies compete for capital	*05_multi_strategy\combine_strategy_score.rts*
Compare against buy & hold / gate on the equity curve	*05_multi_strategy\combine_benchmark.rts*
Add a bull-market filter (and test it honestly)	*06_market_context\regime_filter.rts*
Use market breadth	*06_market_context\breadth_filter.rts*
Limit sector concentration	*06_market_context\sector_limits.rts*
Use VIX / term structure	*06_market_context\vix_filter.rts*
Understand survivorship bias	*06_market_context\index_membership.rts*
Trade futures	*07_futures\futures_trend.rts* (continuous), *futures_term_structure.rts* (individual contracts)
Check if costs kill a futures edge	*07_futures\futures_one_contract.rts*
Optimize parameters	*08_optimization\opt_grid.rts*, then *opt_constraints.rts*
Know whether I'm curve-fitting	*08_optimization\opt_overfitting.rts*, *opt_walk_forward.rts*, *opt_random_baseline.rts*
Get today's signal list	*09_scans_orders\scan_basic.rts*, *scan_multi_filter.rts*
Generate tomorrow's orders	*09_scans_orders\orders_tomorrow.rts*, *orders_basket.rts*, *orders_rebalance.rts*
Analyze my actual fills	*09_scans_orders\track_actual_trades.rts*
Import data (any source)	*10_data_import\import_<source>.rts*
Attach metadata / earnings dates	*10_data_import\syminfo_events.rts*
Join external data to bars	*10_data_import\external_series.rts*
Customize results / trades / graphs output	*11_output_reporting\custom_*.rts*
Compute portfolio-level analytics in-test	*11_output_reporting\testdata_stratdata.rts*
Model deposits, fees, cash interest	*11_output_reporting\accounting.rts*
Trade non-US markets / multiple currencies	*12_special_topics\international_asx.rts*, *multi_currency.rts*

Write reusable formula functions	*12_special_topics\library_functions.rts*
Debug a strategy that "runs but looks wrong"	*12_special_topics\debug_and_validate.rts*

Conventions Used Throughout

- Realistic costs everywhere: *Commission: Max(1, 0.005 * Shares)* (IB-style), slippage or *LimitExtra* where fills would otherwise be optimistic, *Slippage: TickSize* on futures.
- Every portfolio strategy has at least one capacity limit (*MaxPositions*, *MaxExposure* or *MinFreeCash*).
- Test start dates leave a warmup year after the import start so long-lookback indicators are ready, and never precede an index's constituency coverage.
- Each script ends with a small *#ifdef research* include used by AI-assisted research sessions (see **Using RealTest with Agentic AI**). It is inert in normal use.
- Some strategies are deliberately mediocre. When a filter doesn't help or an edge dies to transaction costs, the Notes say so - reading an honest negative result is part of the lesson.

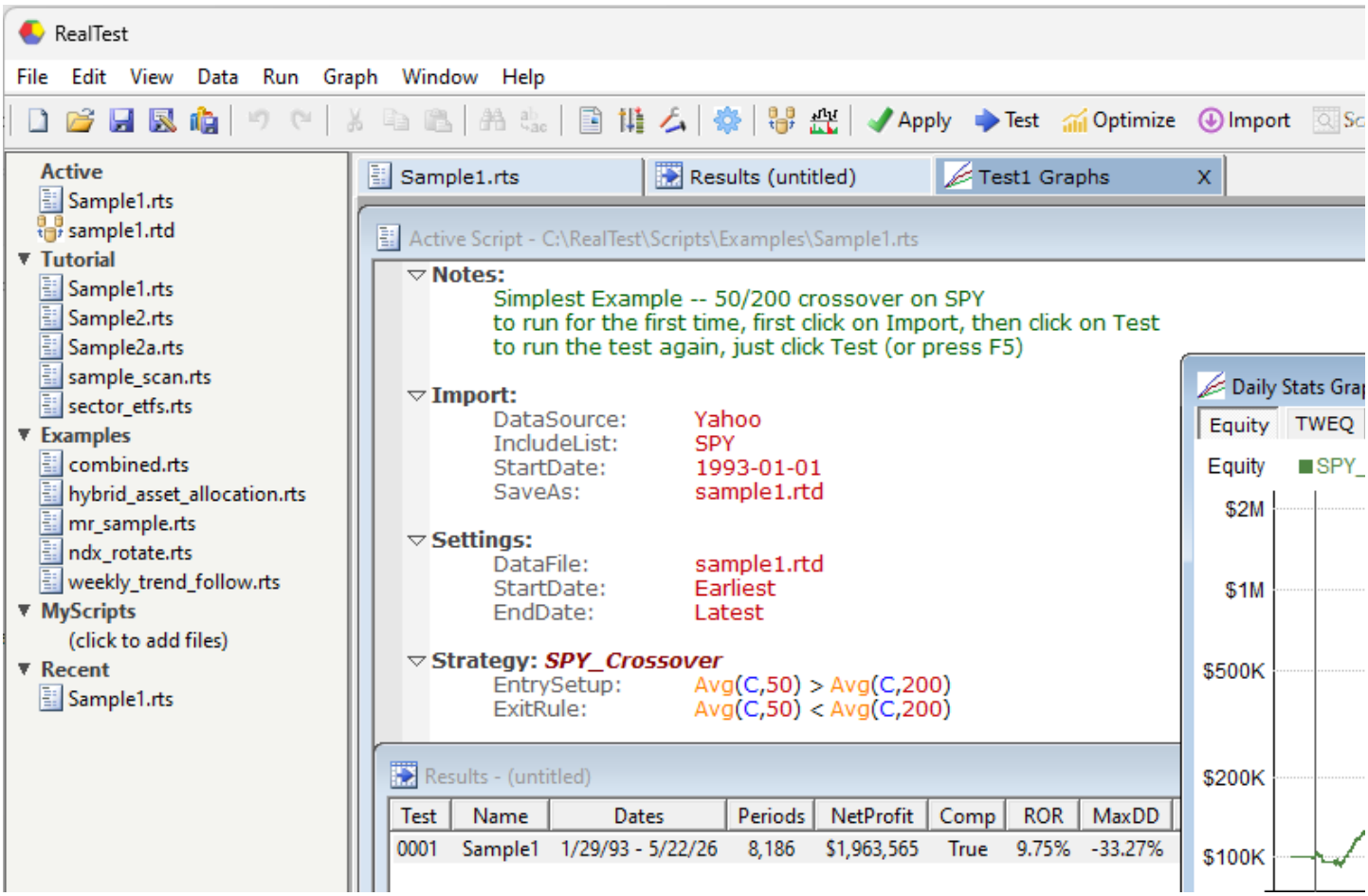
6. Software User Interface

RealTest presents its windows using the classic Windows multiple-document interface. These topics describe each part of that interface.

- **The Main Window** - the child-window layout, window tabs, and docking panels
- **Common Menus** - the main menu bar shared across all windows
- **Child Window Menus** - the menus specific to each type of child window
- **Bars and Panels** - the tool bar, status bar, and Files, Settings, and Debug panels
- **Text Editor Windows** - script and log windows
- **List Windows** - results, trades, scans, and other tabular windows
- **Graphical Windows** - stats graphs, optimization graphs, charts, and trade plots
- **Program Options Dialog** - application-wide settings
- **RealTest Function Key Reference** - the keyboard shortcuts

6.1. The Main Window

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.

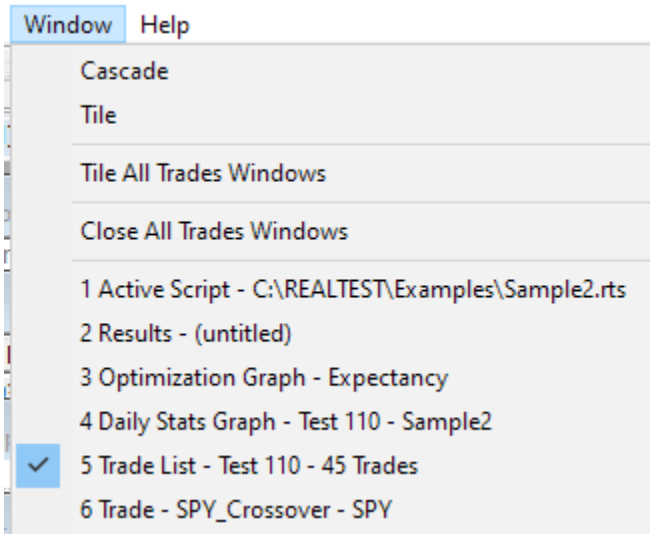
The child-specific menu is also always available as a popup menu by pressing the right mouse button within the child window.

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

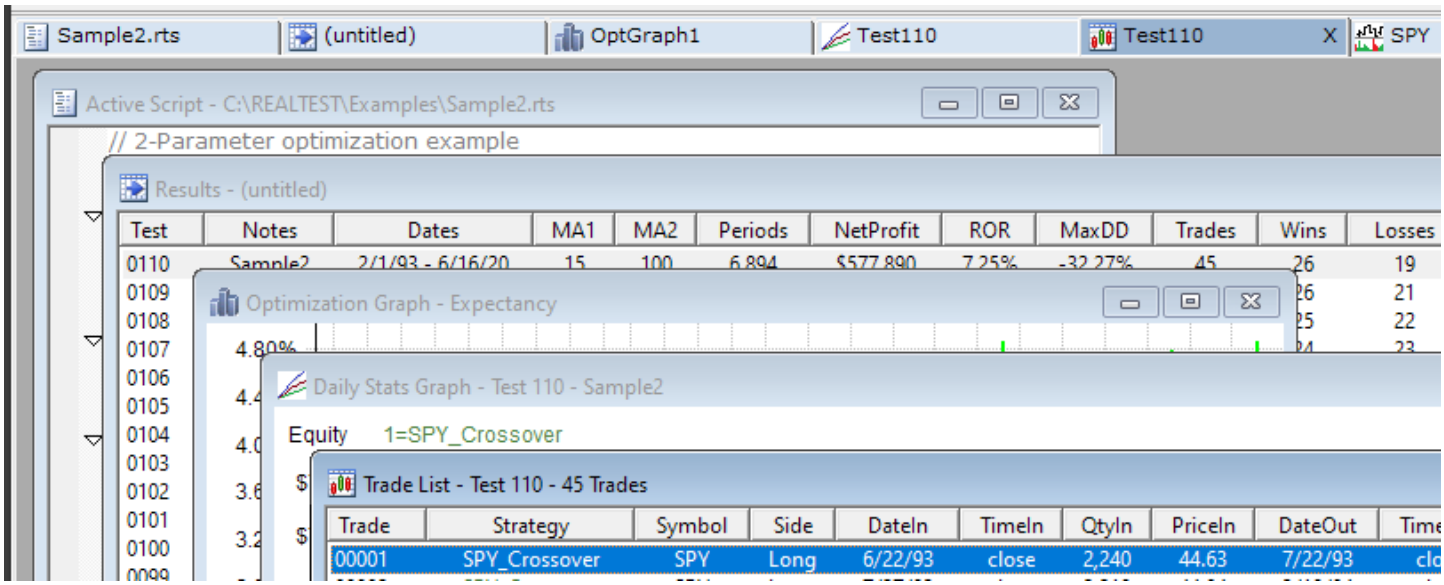
Window menu.

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 three permanent dialog boxes:

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

Each of these dialogs can easily be hidden or shown using the "View" menu and/or dedicated function keys (F10, F11 and F12).

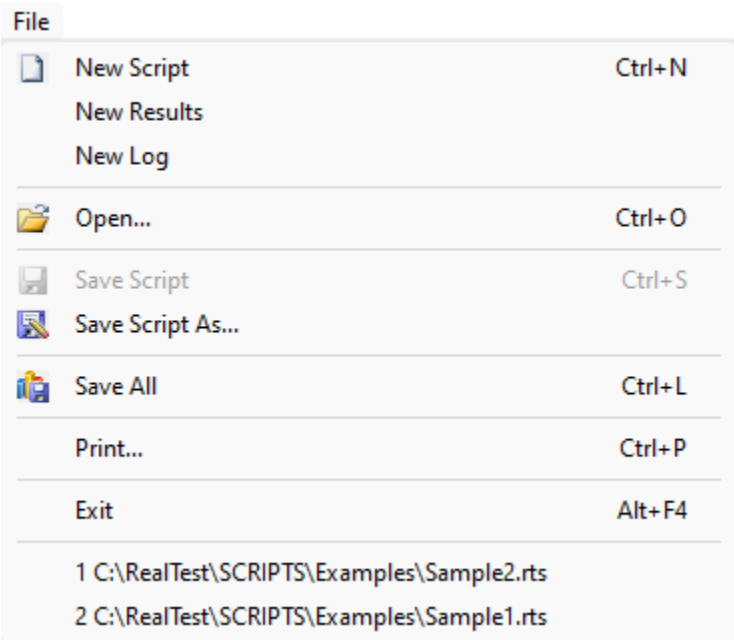
The Window Tabs, Tool Bar and Status Bar can also be optionally hidden or shown using the **View menu**.

6.2. Common Menus

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

6.2.1. File Menu

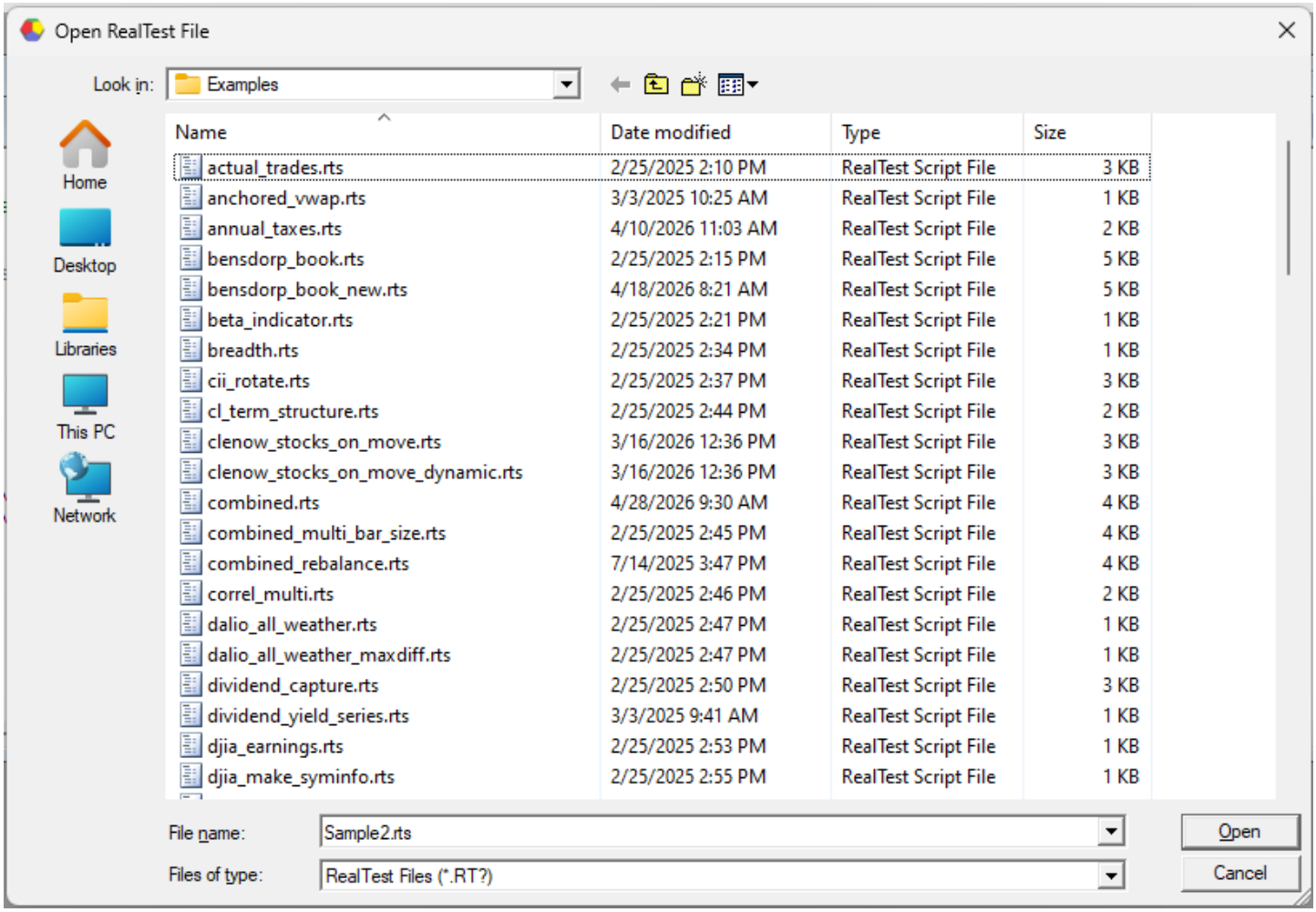
The *File* menu, as in most software, is used to open or save files, or to exit the program.



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

A list of up to 9 of the most recently opened files is automatically shown at the bottom of the menu. It applies the same Recent file-type filter (RTS / RTD / RTR / TXT / LOG / CSV) that you set on the Files tab of **Program Options**, so the menu and the **Files Panel**'s Recent section list the same files (the panel can show up to 100; the menu is always capped at 9). Long paths are shortened so they don't widen the menu.

Selecting *File / Open...* presents a standard Windows file selection dialog. You can select more than one file at once (Ctrl- or Shift-click), and RealTest will open each of them.



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

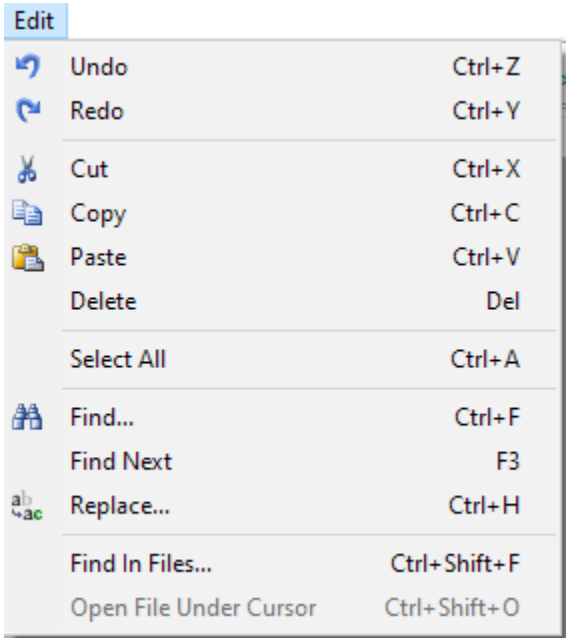
The default file type selection is *RealTest Files (*.RT?)*. To see other kinds of files, change this selection.

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

File Print is only enabled when the active window is a Text Editor Window. Printing is not supported for any other window type.

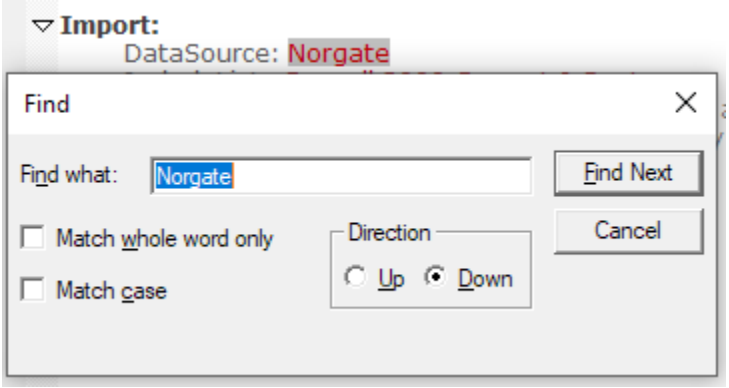
6.2.2. Edit Menu

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



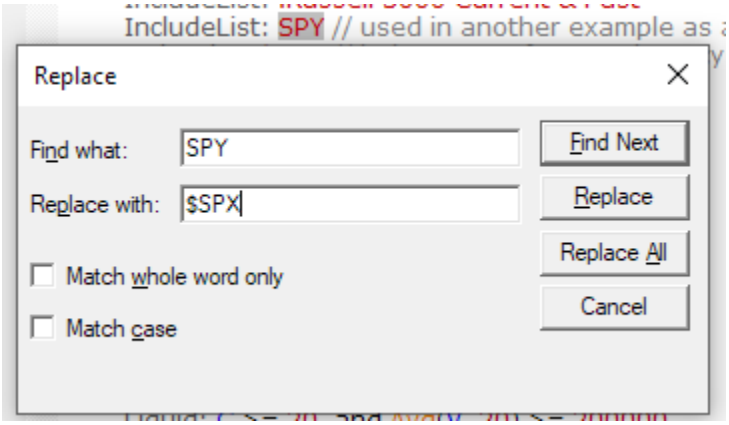
Menu items with icons next to them are also accessible using the Tool Bar.

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

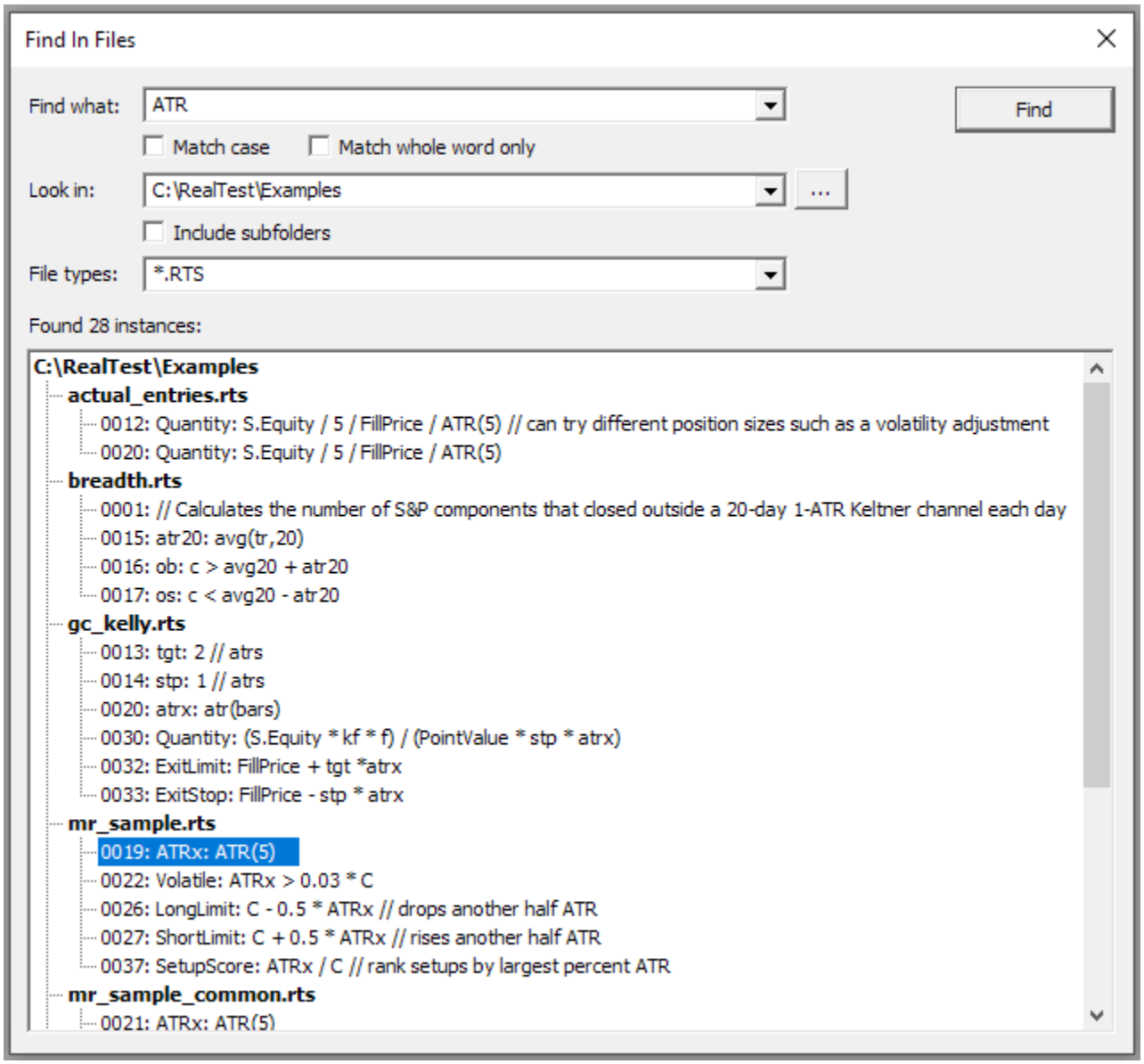


The *Find what* field is automatically filled in with the selected text (or text under the cursor if none is selected) in the editor.

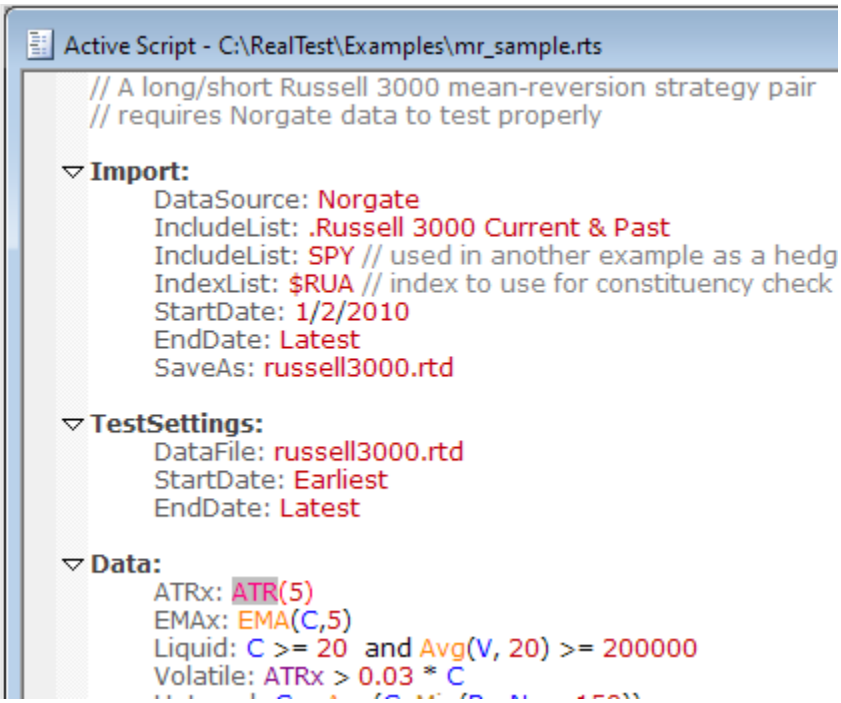
Use *Edit / Replace...* to find specific text and replace it with other text, either once or multiple times:



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

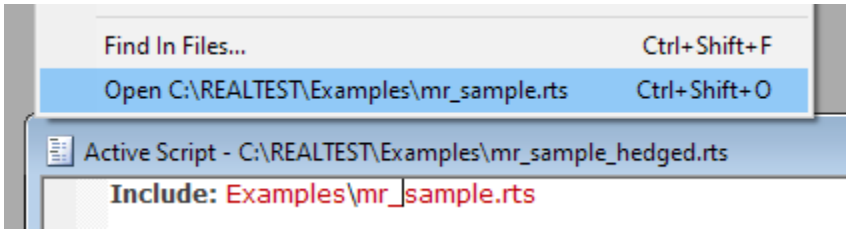


Double-click on any line of found text to open that file and highlight that instance of the text:



The last item on the menu, *Open File Under Cursor*, changes to show the path of the specific file that

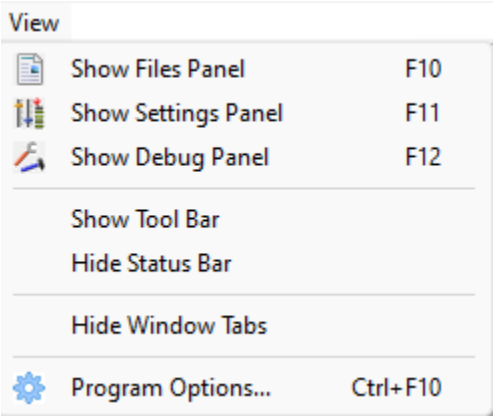
would be opened when the cursor in the active window is within a valid file path:



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*.

6.2.3. View Menu

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



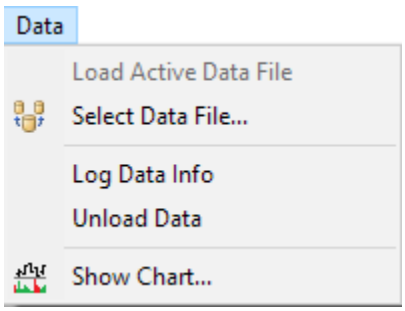
Menu items with icons next to them are also accessible using the Tool Bar.

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

The last item invokes the **Program Options** dialog box.

6.2.4. Data Menu

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

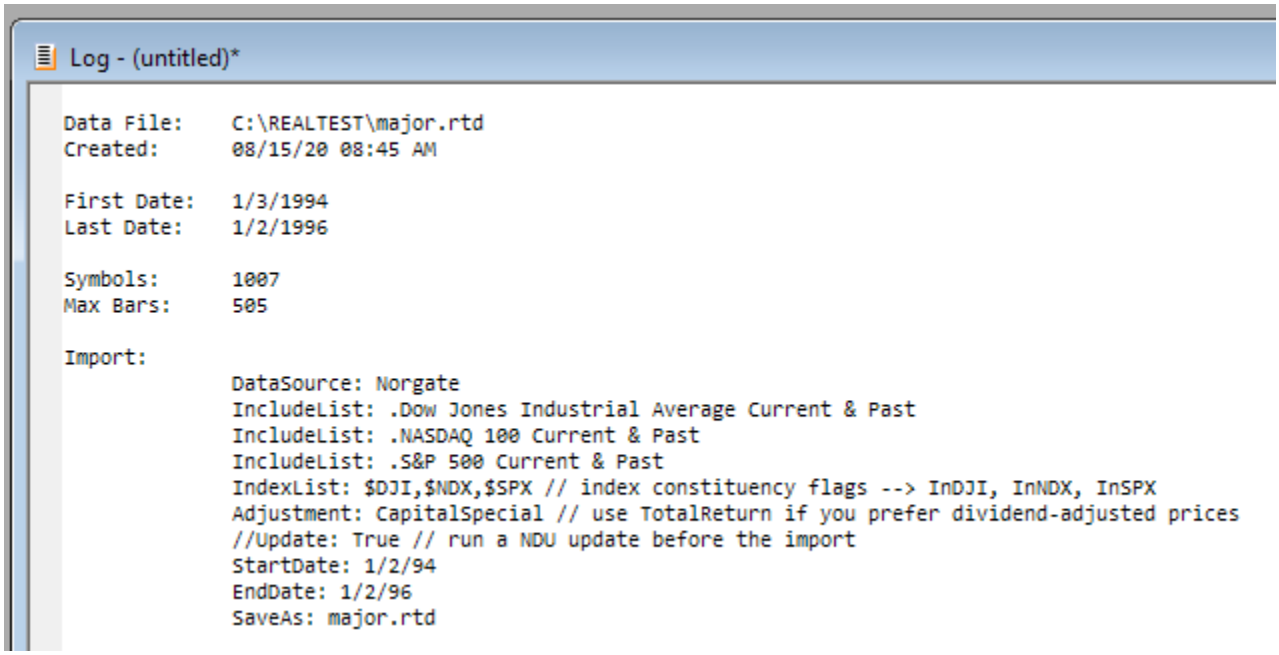


Load Active Data File loads the active data file (.RTD) directly into memory, without a file dialog. It is enabled only when the data is not already loaded.

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.

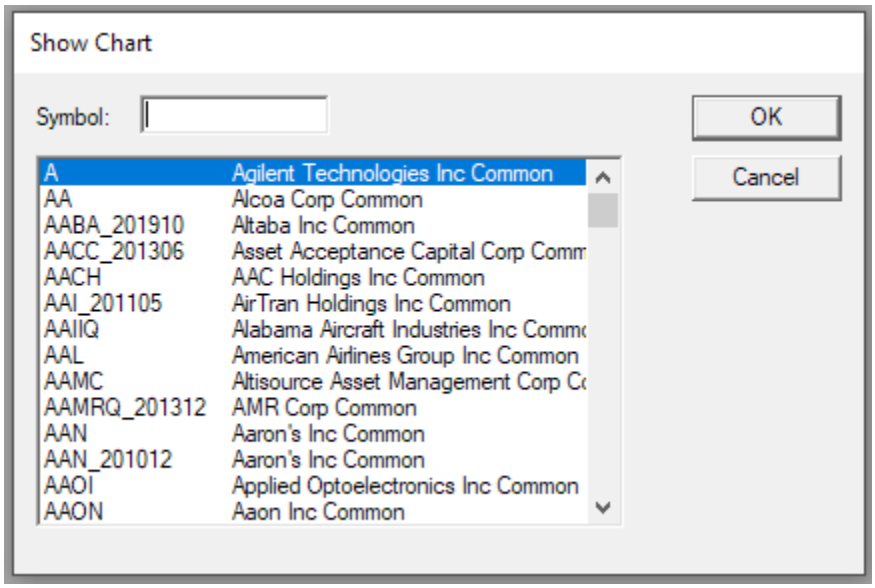
This is an example using the data file created by the **import_norgate.rts** example script:



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.

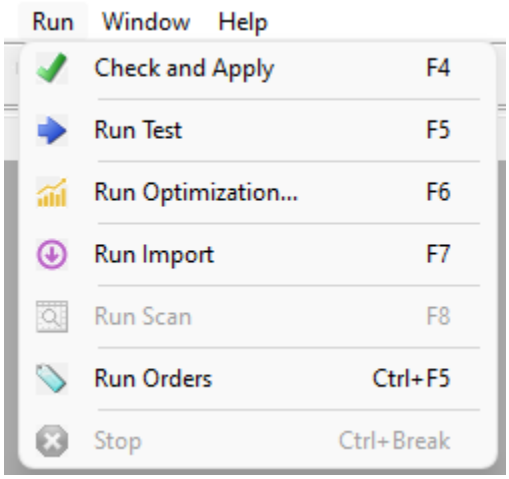
A dialog box is opened with a list of available symbols to choose from:



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**.

6.2.5. 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 **RealTest Optimizer**, 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.

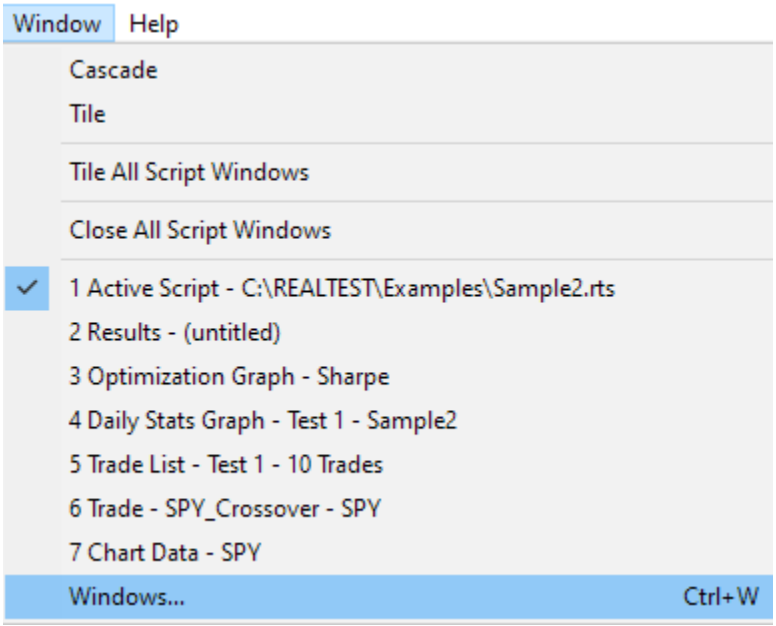
Run Scan runs the **Scan** section of the active script.

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.

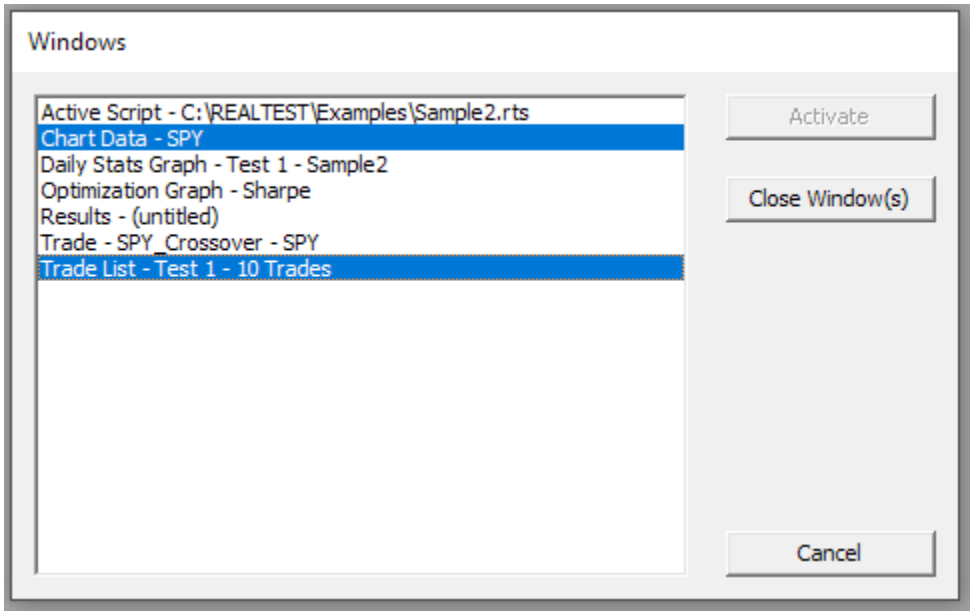
Stop stops whatever operation is currently running.

6.2.6. Window Menu

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

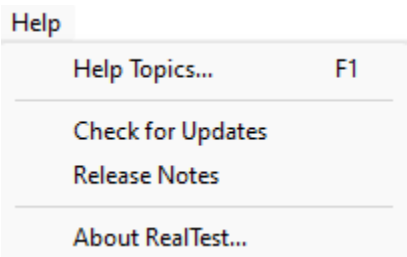


If you have more than 9 windows open and/or want to select several specific windows to close at once, open the *Windows Dialog* from the above menu or by pressing Ctrl+W.



6.2.7. 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.

Check for Updates checks whether a newer program version is available and optionally installs it. *Release Notes* opens the latest release notes in your web browser.

6.3. Child Window Menus

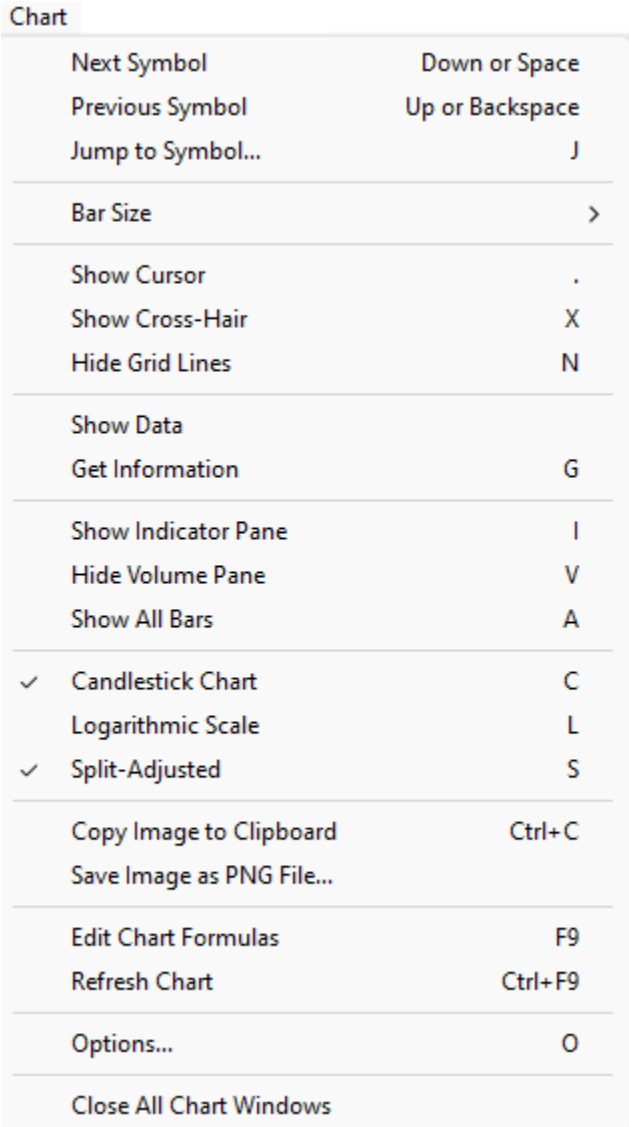
The menu bar items described in this section become visible only when the active child window is the corresponding type. For example, the Chart menu appears in the menu bar when a chart is the active window. These menus can also be accessed by pressing the right mouse button anywhere within the child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

6.3.1. 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.

Bar Size changes the periodicity of bars shown on the chart. Choices are Daily, Weekly, Monthly, Quarterly, Yearly.

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 / Hide Grid Lines toggles the dotted horizontal price lines and vertical date lines on or off. The price and date labels still display when the grid is hidden.

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.

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).

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.

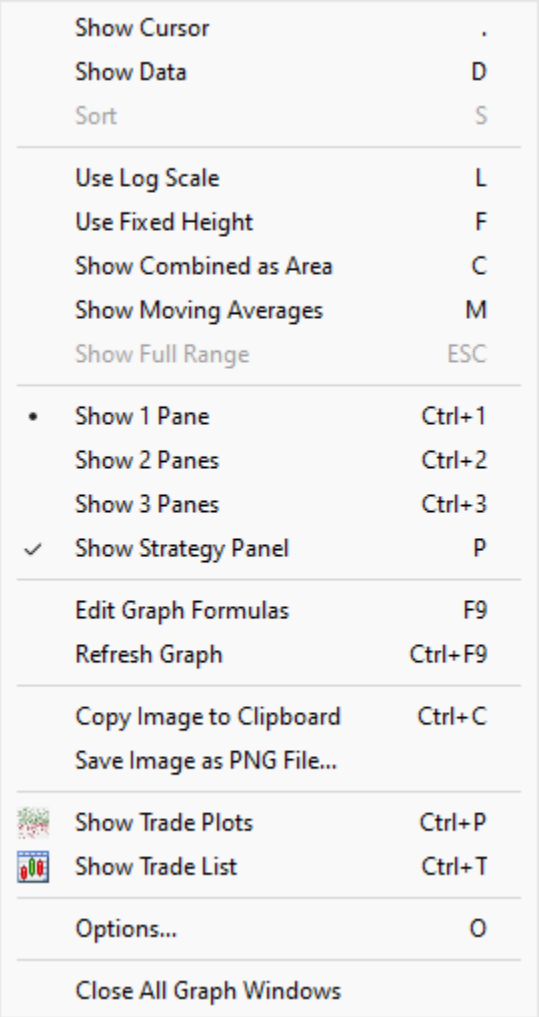
Edit Chart Formulas opens a **script window** with the script that contains the **Charts Section** that was last applied to a chart. Most of the time, this will be *charts.rts* unless you've recently applied a different script with a replacement *Charts* section.

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.

6.3.2. 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.



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. If any value is zero or negative, the toggle silently has no effect and the graph stays arithmetic. 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.

Show 1 / 2 / 3 Panes sets how many stacked graph panes the window contains (also Ctrl+1, 2 and 3). Each pane has its own tab bar and graph selection, while the cursor, zooming and strategy choices stay synchronized across panes. See **Daily Stats Graphs** for details.

Show Strategy Panel (also the P key) toggles the strategy-list side panel, which replaces the in-graph legend with a checkbox list docked to the left of the graph. See **Daily Stats Graphs** for details.

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.

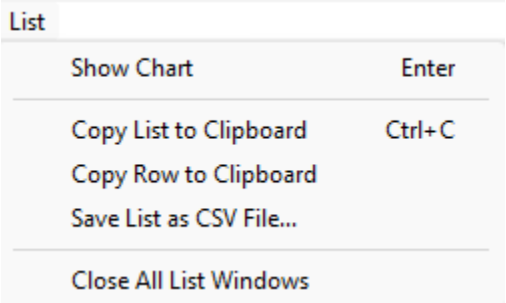
Show Trade Plots opens a **Trade Plot Window** for the test behind the graph. This has the same effect as selecting this item in the **Results Window**.

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.

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

Show Chart opens a candlestick **Chart** of the symbol in the selected row.

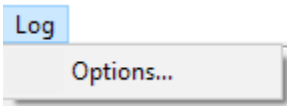
Copy List To Clipboard copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

Copy Row To Clipboard copies only the selected row to the Windows clipboard as tab-delimited text.

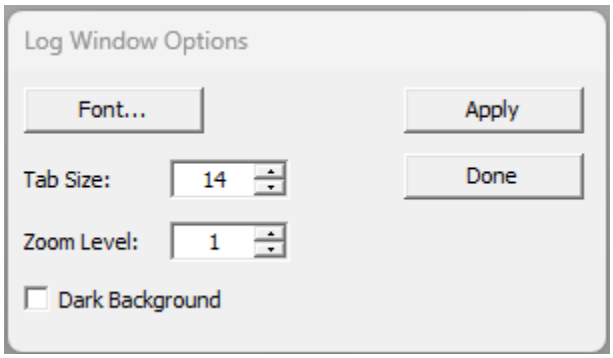
Save List As CSV File prompts for a file path and name, then saves the entire contents of the list to the specified file in comma-delimited text (CSV) format.

6.3.4. 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 default tab size of 14 is recommended for best results when viewing a **Test Summary Report**. Change it here if you prefer something different.

Zoom Level can be used to make text larger or smaller without changing the font size. This can more easily be changed by using Shift + Mouse Wheel.

Check *Dark Background* if you prefer to see white text on a black background in your log windows.

6.3.5. 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.

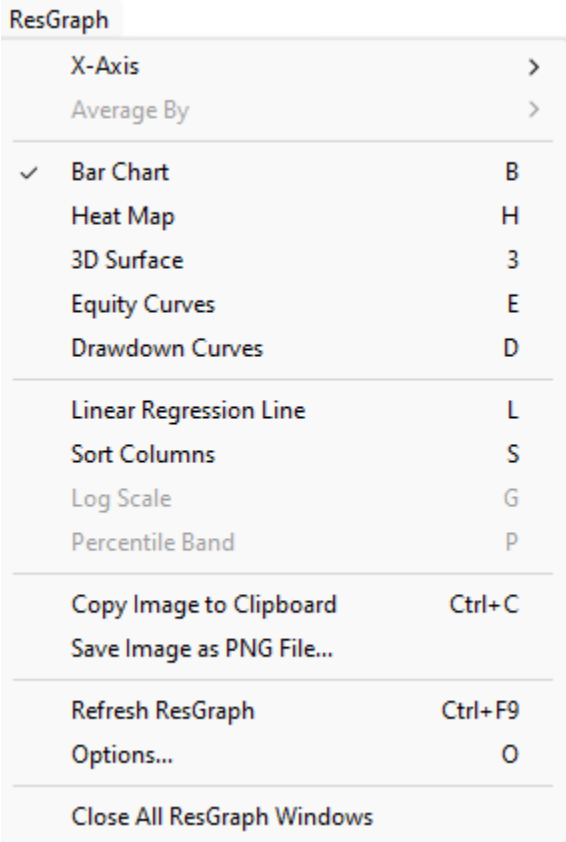
Each item on this menu also has a keyboard shortcut, as shown to the right of the item.

Plot		
✓	Scatter Plot	S
	Profit Distribution	N
	MAE/MFE Distribution	E
	Cumulative Profit	C
	Equal Count Bins	B
	Equal Range Bins	R
	Factor Scan	G
	Monte Carlo Equity	M
	Monte Carlo Equity Drawdowns	D
	Monte Carlo Trades	Shift+M
	Monte Carlo Trade Drawdowns	Shift+D
	Dollars	$
	Trade Percent	%
✓	Alloc Percent	^
✓	Trade Number	T
	Formula	F
	Percentile	I
	Cross Hair	X
	Linear Regression Line	L
	Log Analysis Stats	A
✓	Show Strategy Panel	P
	Copy Image to Clipboard	Ctrl+C
	Save Image as PNG File...	
	Options...	O
	Close All Plot Windows	

See **Trade Plots and Analysis** for details about what all these choices are.

6.3.6. ResGraph Menu

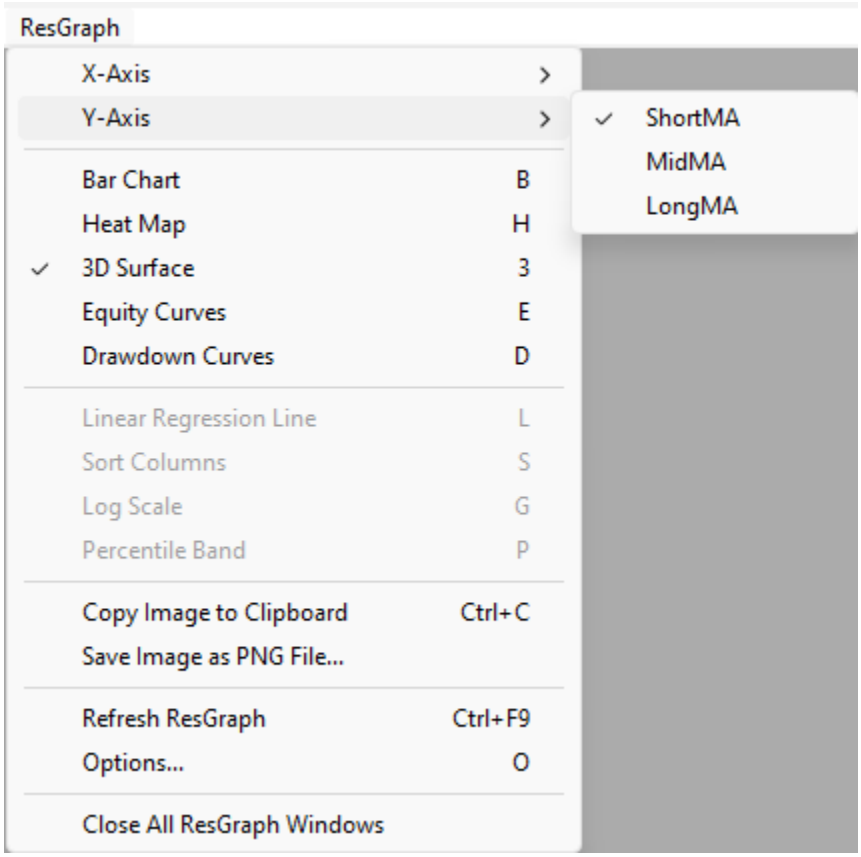
The *ResGraph* menu appears in the menu bar whenever a **Results Graph Window** is active, and can also be accessed by pressing the right mouse button within that window.



This menu has two 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.

The **second sub-menu** also lists the optimization parameters, but its label changes with the graph type. For a *Heat Map* or *3D Surface* it is labeled **Y-Axis**, and the selected parameter becomes the graph's Y axis. For the bar and curve graph types it is instead labeled **Average By**, selecting the parameter to average across (it is disabled when there is nothing to average).



Bar Chart, **Heat Map**, **3D Surface**, **Histogram**, **Equity Curves** and **Drawdown Curves** select which type of optimization graph the window displays (also the B, H, 3, M, E and D keys). Heat Map and 3D

Surface require two or more optimized parameters. See **Optimization Results Graphs** for a full description of each graph type.

Linear Regression Line toggles a least-squares trend line on the graph.

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.

Log Scale toggles logarithmic scaling of the value (Y) axis, most useful for the equity and drawdown curves.

Percentile Band toggles a shaded band that shows the distribution of results across all tests. See **Optimization Results Graphs** for details.

Log Summary Text writes a text summary of the current graph to the Log window. For the Histogram it is a tab-delimited sensitivity table with one row per results column; for the Overfitting Analysis it is the headline metrics plus the walk-forward correlation at every split point. See **Optimization Results Graphs** and **Overfitting Analysis** for details.

Show Values (heat map only) toggles drawing each cell's value inside the cell (also the V key). Values are drawn only when the cells are wide enough to hold them, so a dense grid stays readable.

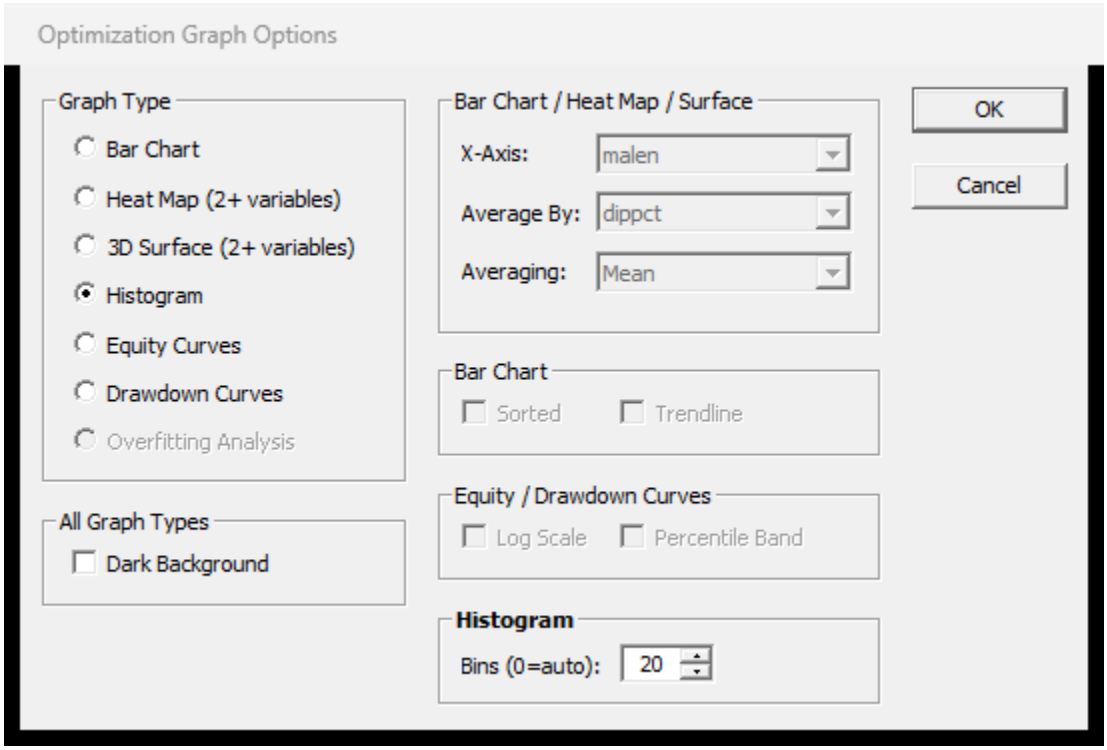
Show Data opens the graph's underlying data as a list window. For the Heat Map it is the grid as a table, one row per Y-axis parameter value and one column per X-axis value, ready to copy into a spreadsheet. For the Overfitting Analysis it is the current view's data points: the per-split selections, or every combination's in-sample and out-of-sample Sharpe at the split being shown.

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.

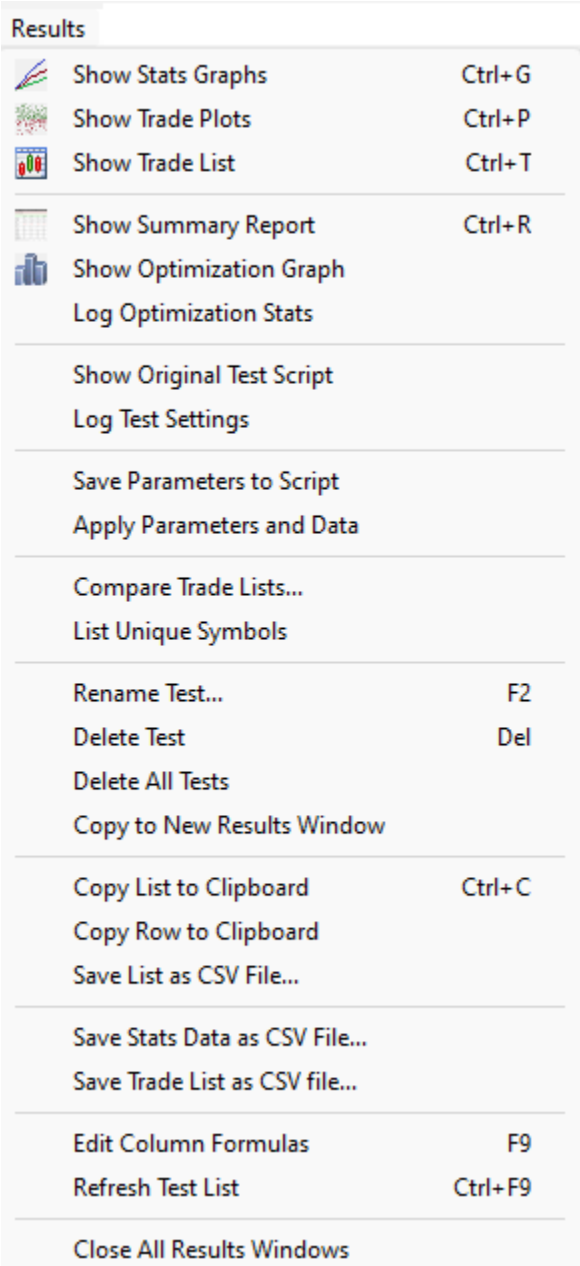
Refresh ResGraph recalculates the optimization data and redraws the graph.

Options opens the Optimization Graph Options dialog, which collects all of the menu's choices in one place, grouped by the graph type they apply to (the groups for the currently selected type are shown in bold), and adds three that are not on the menu: the *Averaging Method*, a Dark Background option, and the histogram's *Bins* count. See **Optimization Results Graphs** for details.



6.3.7. 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 Graphs opens a **Daily Stats Graph Window** for each selected test. This can also be done by double-clicking on a test row or selecting one or more rows and pressing Enter.

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

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

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

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

Save Parameters to Script makes the selected test's parameter values the defaults in the active

script. It is enabled when exactly one row with parameter values is selected, and also works for rows loaded from a saved results (RTR) file. The active script is checked (parsed) first, and a confirmation is requested if the selected test was produced by a different script. Each **Parameters** line is then edited in place: an existing *def* value is updated, a range or list without one gains an explicit *def* clause, and a parameter defined as a single fixed value keeps that value and gains a *def* clause when the test used a different value. The changes are made in the script editor window (left unsaved) and can be reverted with Undo. (This command was named Apply Parameters to Script before 2.0.32.5.)

Apply Parameters and Data puts the whole program in the selected test's context without editing any script file: the test's script becomes the active script (opened if necessary), its parameter values are applied in memory, and its data is loaded and its Data items recalculated with those values as needed. Trade lists, chart indicators, graphs, plots and the Debug panel then show the selected test's own values, which is especially useful after an optimization, when calculated Data items would otherwise reflect the script's default **Parameters** rather than each test's. It is enabled whenever exactly one row is selected, with or without parameters, and also works for rows loaded from a saved results (RTR) file. Nothing is saved: the next run or apply reverts to the script's own parameter values - including an Apply done to pick up edits to the script itself, such as changes to its Trades section. When refining trade list columns in a test's context, simply use Apply Parameters and Data again (with the test still selected) instead of Apply: it re-reads the script and re-applies the test's values in one step. Not available for walk-forward tests, whose parameter values vary by date within the 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 tests from the *Results Window*. This cannot be undone.

Delete All Tests clears the current *Results Window*.

Copy to New Results Window creates a new untitled Results Window and copies the selected tests to it.

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.

Copy Row to Clipboard same as above, but copies the selected rows only.

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).

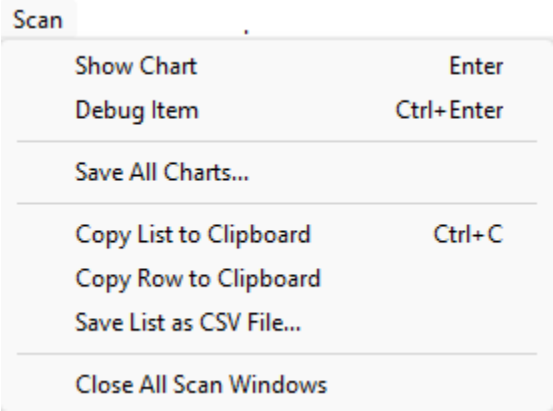
Save Trade List as CSV File prompts for a file path and name, then all standard and custom trade items for every trade in the test are written in CSV columns as raw values (format codes are ignored).

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.

6.3.8. 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 use the size and display options of the most recently viewed chart window. This can also be done automatically by adding **SaveScanChartsTo** to the **ScanSettings** in a script; the companion **SaveScanChartsMax** setting caps how many charts are saved.

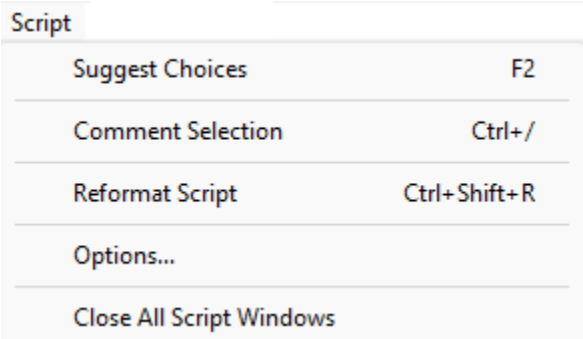
Copy List To Clipboard copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

Copy Row To Clipboard copies only the selected row to the Windows clipboard as tab-delimited text.

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.

6.3.9. 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.

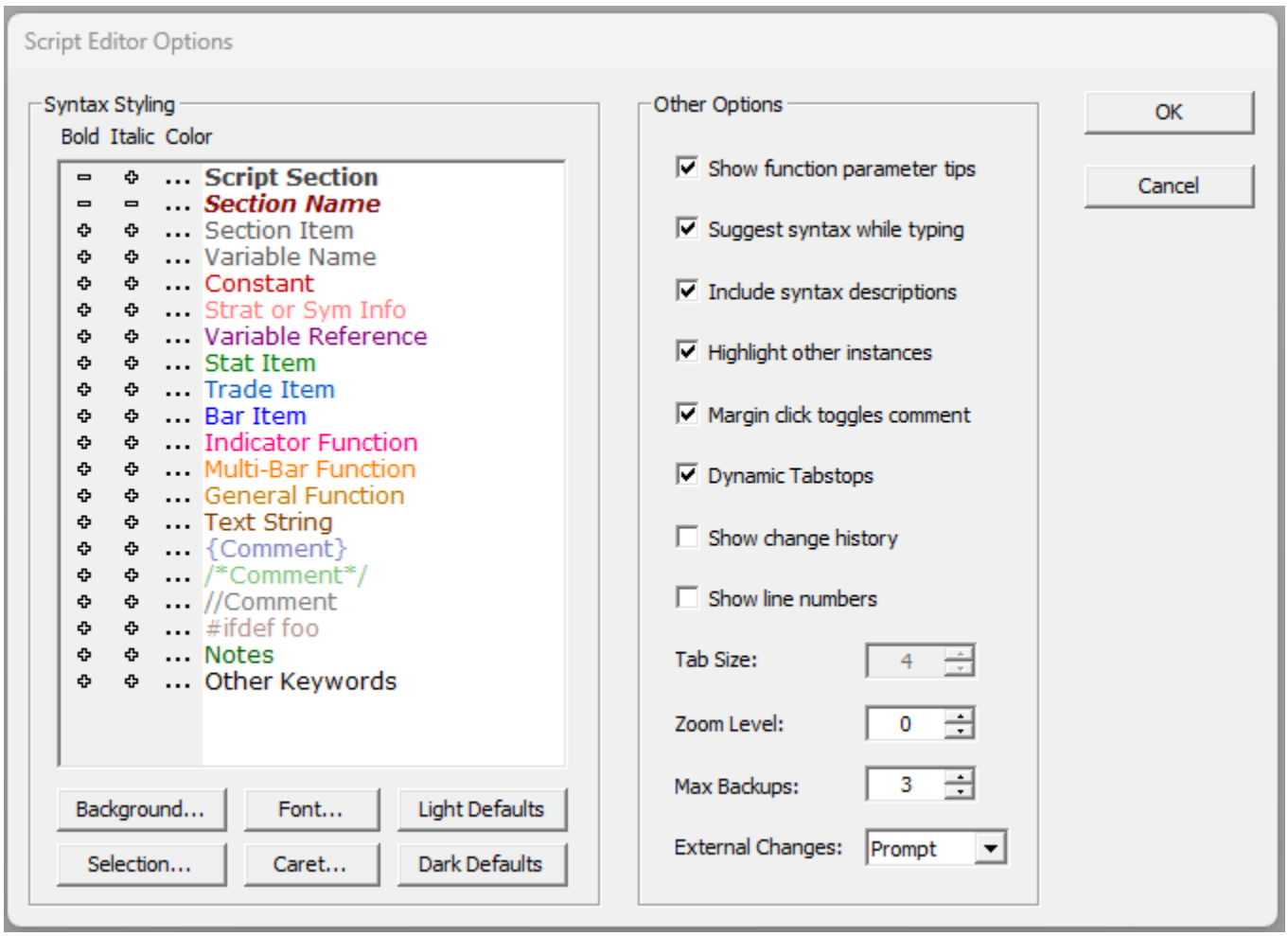
Comment / Uncomment Selection toggles the commented state of the current line or selected set of lines. This is done by either inserting or removing // comment marks from the start of each line. If the cursor is on a **Script Section** definition line when this item is selected, the commenting of the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

Reformat Script reformats the entire script, adjusting indentation and alignment for readability.

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, *Selection* for the selected-text background color, *Caret* to change the color of the blinking cursor, or *Font* to change the typeface or size.

Click *Light Defaults* or *Dark Defaults* to reset all color choices to one of the two default sets of colors.

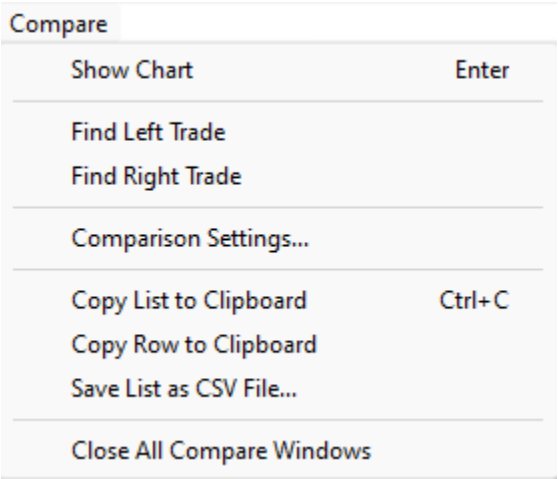
Other options include:

- 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
- whether to show an additional thin margin line that shows which lines have changed while you edit
- whether to show line numbers in the leftmost margin of the script window (clicking the margin still behaves as chosen above)
- the fixed tab size when not using dynamic tabstops
- the current font zoom level (can also be modified using the Ctrl+mouse wheel)
- the maximum number of older versions of the script to keep copies of in the Backups folder
- what to do when a script file is changed by another program

6.3.10. 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.

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

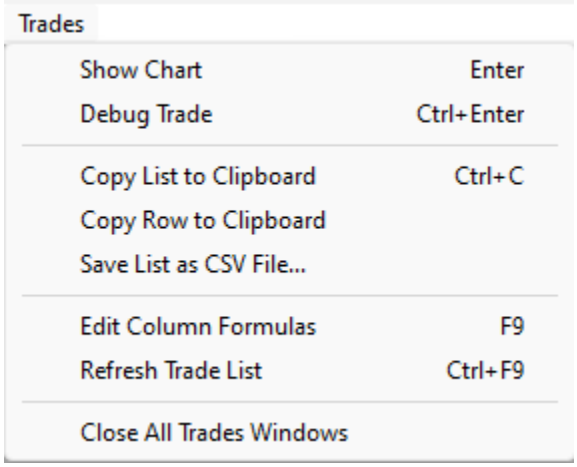
Copy List To Clipboard copies the entire contents of the list to the Windows clipboard as tab-delimited columnar text. This makes it easy to paste the data from the list into a program such as Excel.

Copy Row To Clipboard copies only the selected row to the Windows clipboard as tab-delimited text.

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.

6.3.11. 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.

Save All Charts prompts for a folder path, then saves a chart image for every trade in the list (as currently filtered and sorted) to that folder as *symbol_entrydate_tradenumber.PNG*. Existing contents are not deleted first, but files with identical names will be replaced without asking. Saved charts use the size and display options of the most recently viewed chart window. This can also be done automatically at the end of every test or orders run by adding **SaveTradeChartsTo** to the script's Settings.

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.

Copy Row To Clipboard copies only the selected row to the Windows clipboard as tab-delimited text.

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.

6.4. Bars and Panels

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

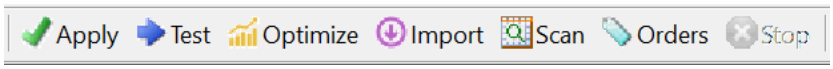
6.4.1. Tool Bar

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

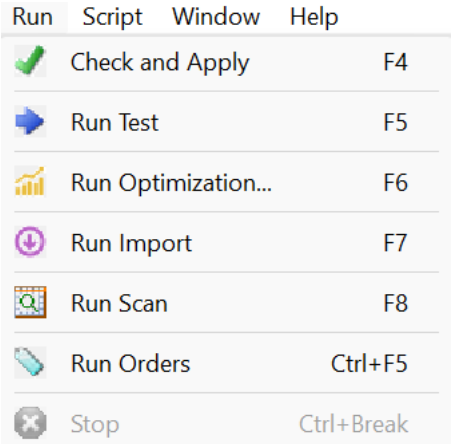
- Check and Apply (F4)
- Run Test (F5)
- Run Optimization (F6)
- Run Import (F7)
- Run Scan (F8)
- Run Orders (Ctrl+F5)
- Stop (Ctrl+Break)

These commands can be accessed in multiple ways:

- From the **Tool Bar**:



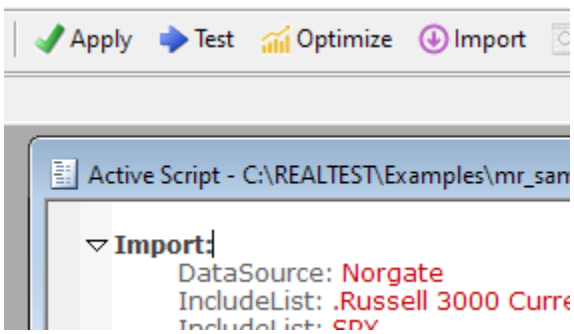
- From the **Run Menu**



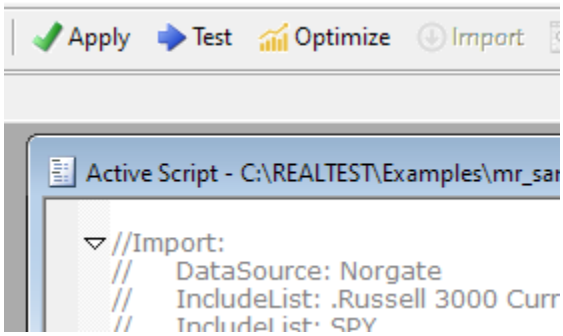
- By using function keys as shown above

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.

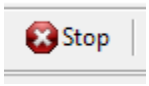


VS

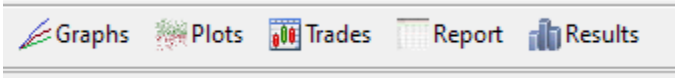


The run commands for the active script remain accessible even when a non-script window (results, graph, chart, etc.) is in the foreground.

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.

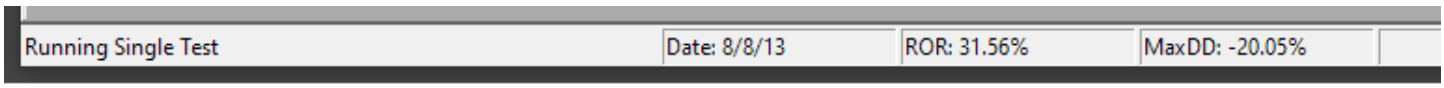


After a test is run, the Tool Bar buttons that invoke **Results Analysis** features are enabled.



6.4.2. 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:

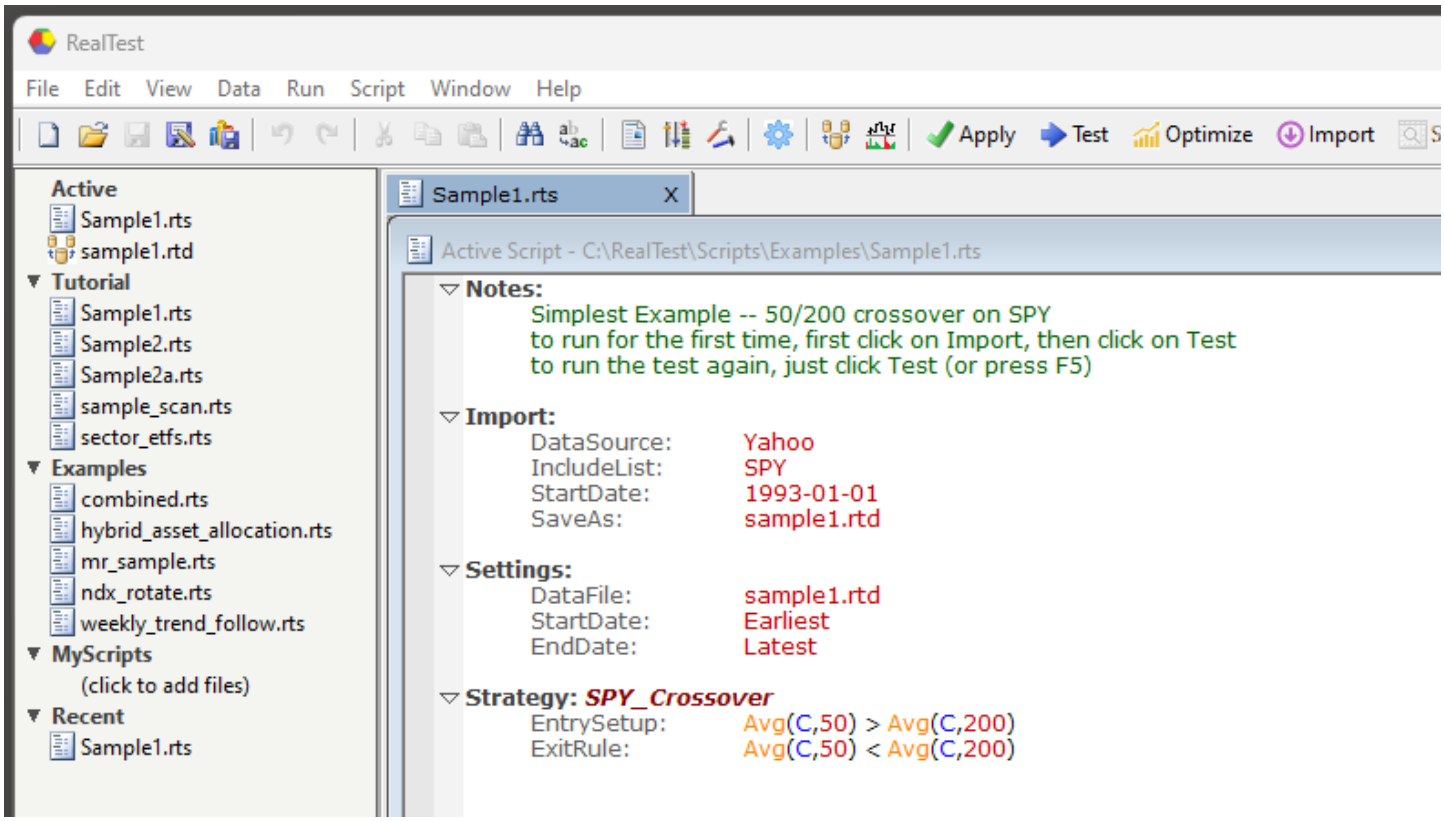
```
Active Script - C:\RealTest\Results.rts
Notes:    this is the default set of columns for Results windows  {...}

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 wa

          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.

The status bar also includes a **Memory** field showing current memory load as two percentages, e.g. *Memory: 10%/55%*. The first number is the share of the computer's RAM in use by this RealTest instance; the second is the total in use by all running programs.

6.4.3. Files Panel

The **Files Panel** is a vertical sidebar on the left edge of the main window that provides quick one-click access to the RealTest files you use most.



It can hold any number of user-curated named **Lists** of files (scripts, data files, results files, logs, CSVs, etc.) above an auto-maintained **Recent** files section.

An **Active** section at the top shows the currently loaded data file and the active script.

This panel can be hidden or shown using the **View Menu** or by pressing the F10 key. The splitter on its right edge can be dragged to adjust its width.

The **Files Panel** includes the following sections:

- **Active**

  Shows the data file currently loaded in memory and the script designated as Active Script.

  It is informational only and cannot be reordered or right-clicked.

- **Lists**

  Any number of user-defined named groups of files. Each list can hold up to 100 files.

  On first launch RealTest creates two default lists, *Tutorial* and *Examples*, pre-populated with some representative sample scripts. An empty *MyScripts* list is also included as a starting place for your own files.

  Files can be added to a list by right-clicking the list's header (or its "(click to add files)" placeholder) and choosing *Add Files...*, or by dragging files in from the Recent section or from another list.

  List entries can be reordered within a list by dragging, and the lists themselves can be reordered using *Move List Up* and *Move List Down* on the header right-click menu.

- **Recent**

  Auto-maintained list of recently opened files (the same set shown on the **File Menu**'s Recent submenu).

  Files in this section can be removed individually or dragged into a list to make them permanent.

  How many recent files are shown (1 to 100) and which file types appear are set on the Files tab of **Program Options**.

Click a section header to collapse or expand that section. Double-click a file to open it. Hover over any row to see its full path.

Right-clicking a file row offers the following actions:

- **Open** - opens the selected file (or all selected files if multiple are selected)
- **Open Containing Folder** - opens the file's folder in Windows Explorer with the file selected
- **Copy Path** - copies the full path(s) to the clipboard
- **Add to List** - adds the selected file(s) to another named list (submenu)
- **Remove from** *list-name* - removes the selected file(s) from the list they belong to
- **Remove from Recent** - removes the selected file(s) from the Recent section

Right-clicking a list's header offers list-management actions:

- **Remove** *list-name* **From Panel** - removes the list (the underlying files are not deleted)
- **Rename List...** - renames the list
- **Move List Up** / **Move List Down** - reorders the lists within the panel
- **Add Files...** - opens a file picker to add one or more files to this list
- **New List...** - creates a new named list

Dragging a file within a list reorders it. Dragging a file from one list to another moves it. Dragging a file from the Recent section into a list adds it to that list (the Recent entry is left alone).

The panel's state -- list names, list contents, list order, and which sections are collapsed -- is saved in *filespanel.ini*, located in the same folder as the active *realtest.ini*. This file may be edited directly; relative paths in the file are resolved against the user's script folder so install-relative entries remain portable.

The Files tab of the **Program Options** dialog also offers an option to place the Settings and Debug panels beside the Files panel at full height. The panel's background color can be changed on the Display tab of the same dialog.

6.4.4. Settings Panel

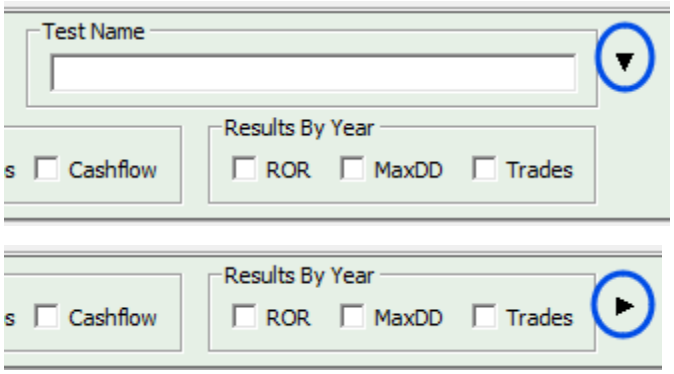
The **Settings Panel** supplies the key run parameters and output selections for the next test or scan that are not set **by the script** itself.

Any setting that a script specifies always overrides the panel for that run.

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

The panel has two rows. The top row holds the **run parameters** -- Date Range, Bar Size, Account Size, Currency, and Test Name. The bottom row holds the **output options** -- Test Output, Keep Trades, Log Detail, and Results By Year.

The **chevron** at the top-right corner collapses or expands the top row. Click it to hide the parameters row (the chevron points down when the row is expanded and right when it is collapsed, matching the **Files Panel**). With the top row collapsed, the output row slides up and the panel's height is halved. The collapsed or expanded state is remembered between sessions.



How the run parameters are applied

The top-row parameters fill in any of these settings that your script does not specify -- but only while the parameters row is actually visible and editable.

Whenever the parameters row is **not** an interactive surface -- the panel is hidden, the top row is collapsed with the chevron, or the script is run from the command line (batch mode) -- RealTest ignores the saved panel values for these parameters and uses its own built-in defaults instead. The run is then determined entirely by the script plus RealTest's standard defaults, rather than by whatever values happen to be left in the panel.

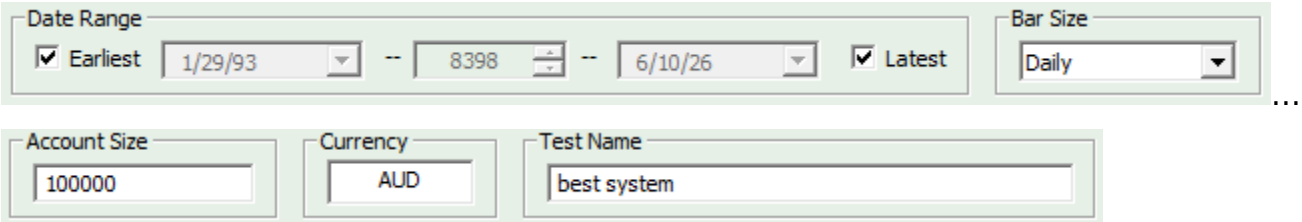
Your saved panel values are preserved in this case: they are restored as soon as the run finishes, so a hidden, collapsed, or batch run never overwrites them.

In effect, collapsing the top row has the same impact on a run as hiding the entire panel -- it switches these parameters from "use my saved panel values" to "use the standard defaults." Expand the row (or show the panel) whenever you want the panel's Date Range, Bar Size, Account Size, Currency, or Test Name to apply to the run.

A script directive always takes precedence over both the panel and the defaults, and only for that run -- for example, a script's **Currency** or date-range setting is rolled back afterward, leaving the panel unchanged.

The output options on the bottom row are **always** applied from the panel, regardless of the chevron, panel visibility, or batch mode.

Run Parameters



- **Date Range**

  Specifies the range of dates to use for the next test or scan. Check **Earliest** to begin at the earliest available date, and/or **Latest** to run through the latest available date; otherwise choose explicit start and end dates. The field between the two dates shows the number of bars in the range, and can be typed into to set the range by length.

- **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, Monthly, Quarterly and Yearly. Sizes above Daily are derived from daily bars. RealTest does not support direct import of larger bar sizes. Can be overridden by the **BarSize** setting in a script.

- **Account Size**

  Specifies the amount of money to use in the simulated account when running tests. Can be overridden by the **AccountSize** setting in a script.

- **Currency**

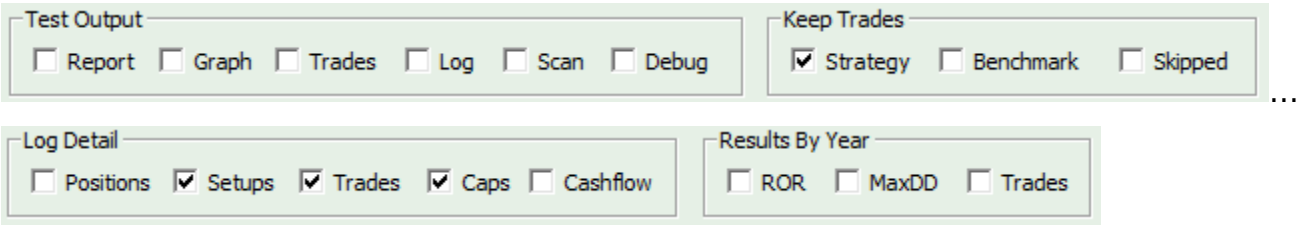
  Sets the base currency of the simulated account, as a three-letter code. Can be overridden by a script's **Currency** directive; the program-wide default is set on the Display tab of **Program Options**.

- **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. Can be set by the **TestName** setting in a script.

Output Options

These selections are also available from the **Outputs** button on the toolbar; the two stay in sync.



...

❖ **Test Output**

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

Some of these (Report, Graph, Trades) can also be run manually at any time by selecting a row in **Results** and then clicking the corresponding toolbar button. The others (Log, Scan, 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
- **Graph** - opens a stats graph at the start of the test and updates it dynamically as the test runs
- **Trades** - displays the round-trip trade list at the end of the test
- **Log** - generates a detailed transaction and position log and displays it at the end of the test (use the Log Detail group below to choose what it contains)
- **Scan** - runs a **TestScan** simultaneously with the test, allowing fully customizable output of test-related data
- **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

❖ **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)

❖ **Log Detail**

Selects which categories of detail are included when a Log is generated (whether from the Log box above or by a script): Positions, Setups, Trades, Caps, and Cashflow. Corresponds to the **LogDetail** setting.

❖ **Results By Year**

Adds per-year columns to the test's row in the **Results window**: ROR, MaxDD, and Trades. Corresponds to the **ResultsByYear** setting.

6.4.5. 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.

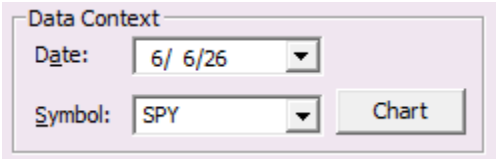
When accessed at the end of a test run with *Debug* selected for **TestOutput**, the full strategy formula context is also available for examination.

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

This panel can be hidden or shown using the **View Menu** or by pressing the F12 key.

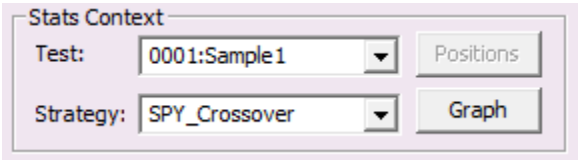
The **Debug Panel** includes the following items:

- **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.

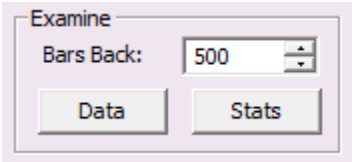
- **Stats Context**



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.

This group also includes a Positions button, enabled only when the last test that was run left open positions; pressing it logs those open positions. (Those positions also appear as "Open" rows in the trade list -- see **EndOfTestExits**.)

- **Examine**



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

- **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":

Log - (untitled)*

```
-----------------------------------------
Symbol:        SPY
Date:          6/15/20
Formula:       C > Avg(C,5)
Evaluation:
               Close = 307.05
               Close[4] = 320.79
               Close[3] = 319
               Close[2] = 300.61
               Close[1] = 304.21
               Close = 307.05
               Avg(307.05, 5) = 310.33
               (307.05 > 310.33) = 0

Result:        0 (0)
```

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 "Debug" checked in **Test Output** 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
```

6.5. Text Editor Windows

RealTest integrates the wonderful open-source Scintilla text editor for use in child windows of two types: **Script Windows** and **Log Windows**.

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.

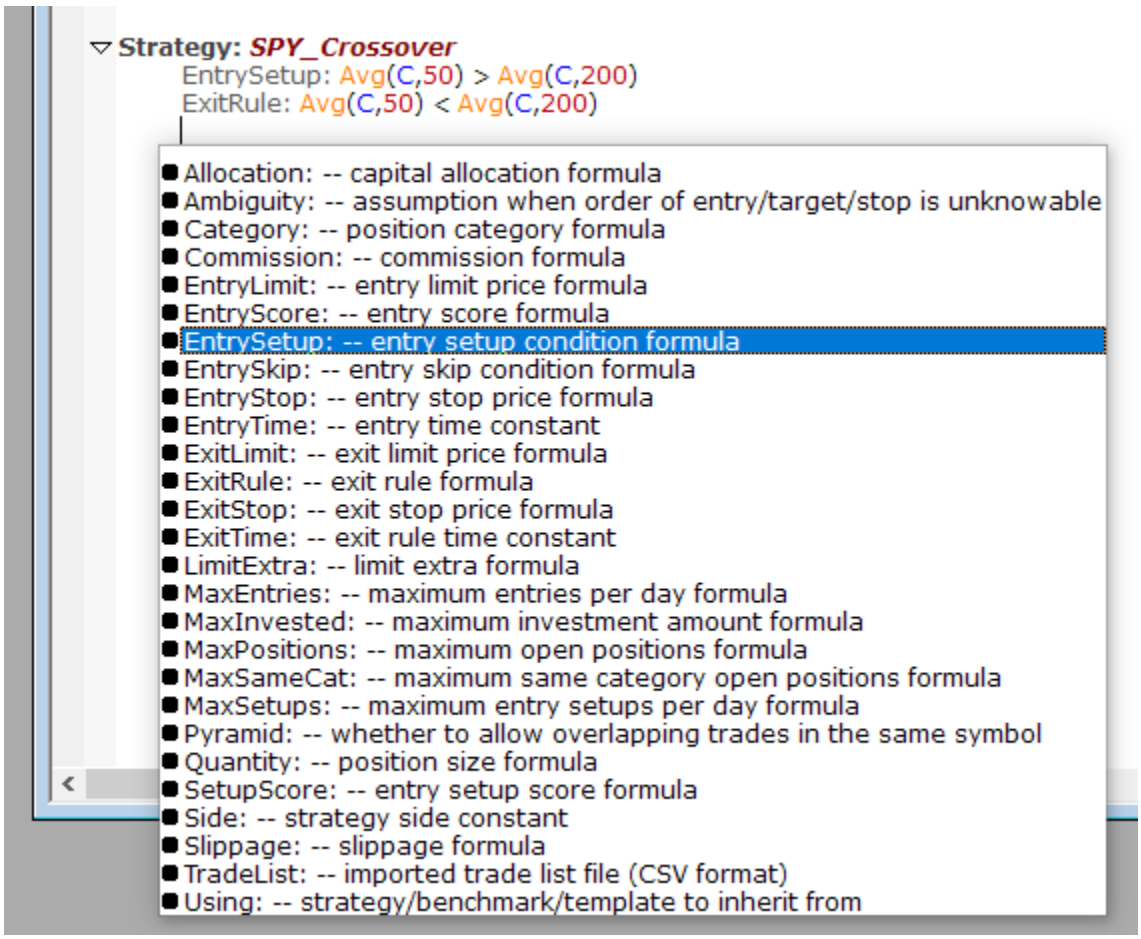
NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER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

6.5.1. 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.

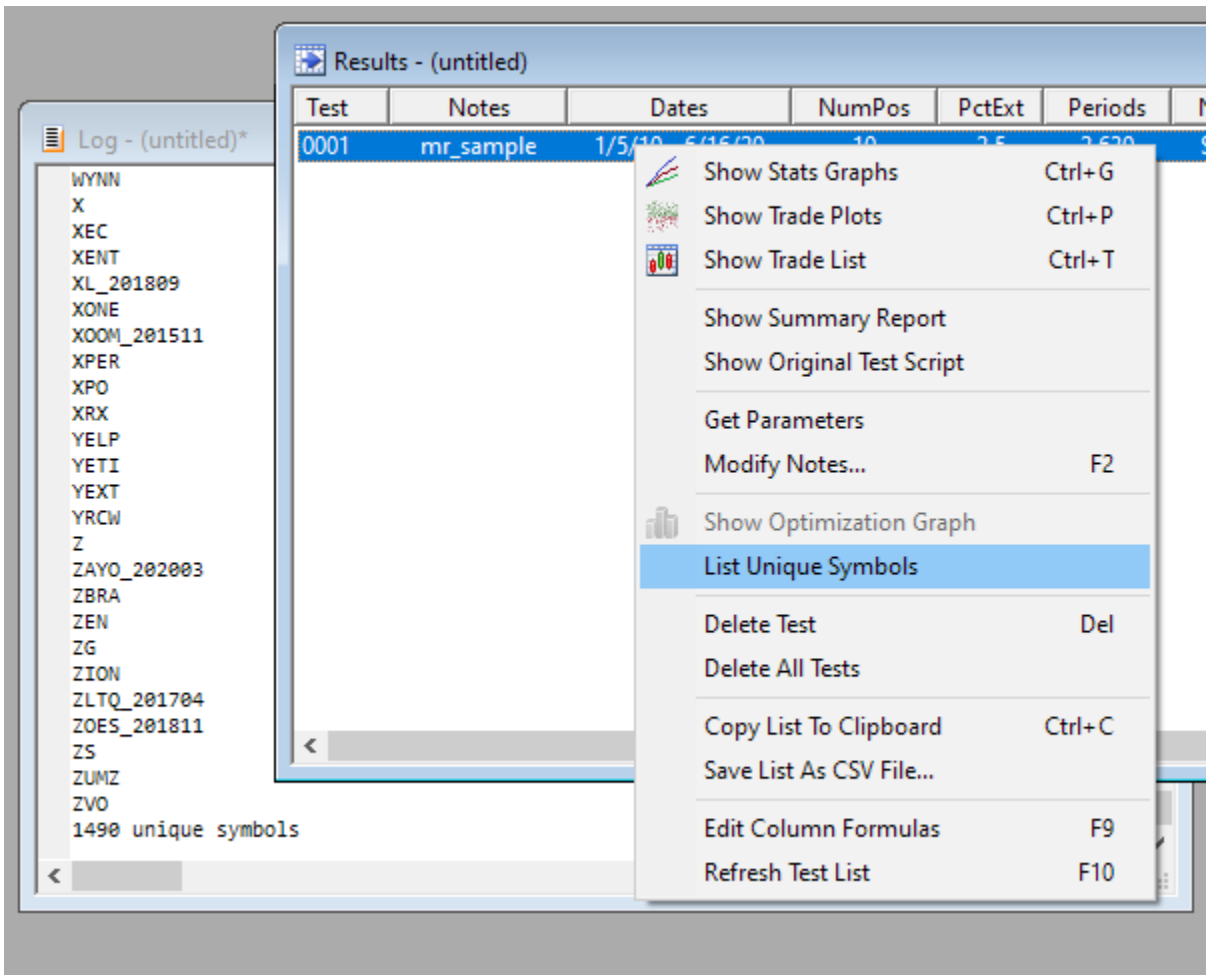
See the **Script Menu** documentation for more about what can be done in a Script Window.

6.5.2. 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.

6.6. List Windows

List windows display tabular data in rows and columns. These topics describe the features they share and each specific type.

- **Common Features** - sorting, copying, saving, and drilling down to detail
- **Results Windows** - one row per test, with summary statistics
- **Trade List Windows** - the individual trades from a test

- **Scan Windows** - date and symbol output from a scan
- **Trade Comparison Windows** - side-by-side comparison of two trade lists
- **Other List Windows** - general-purpose tabular displays

6.6.1. Common Features

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.

Results - (untitled)

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.

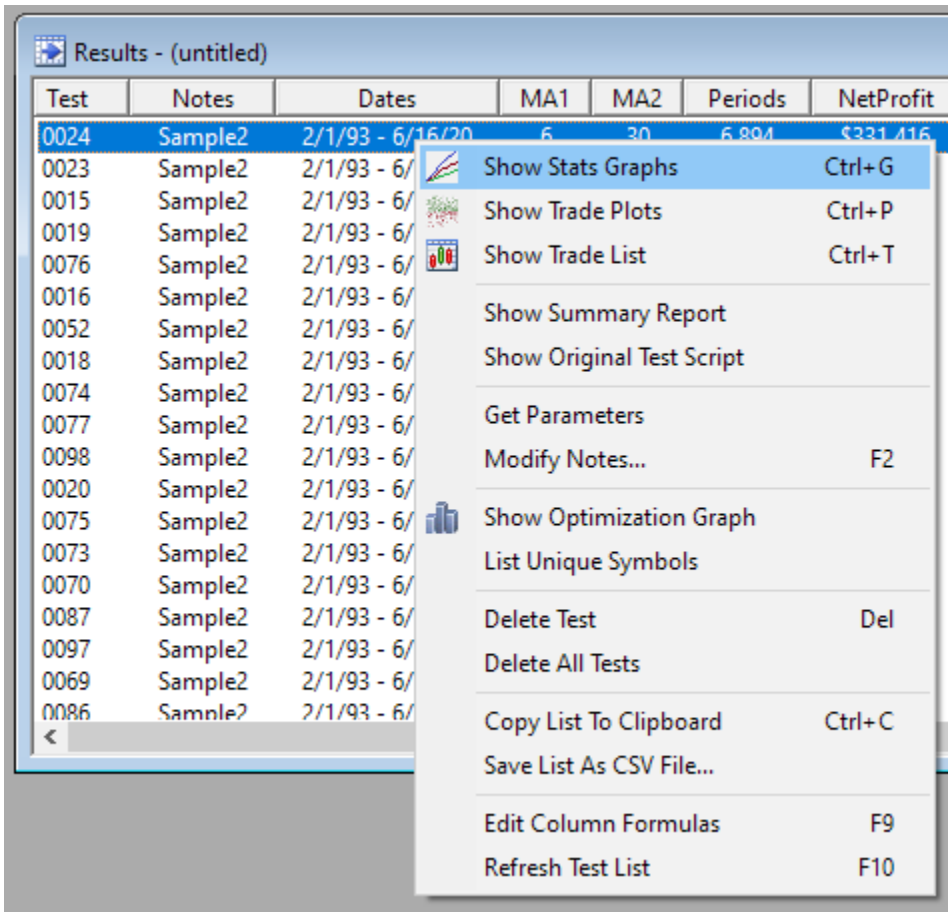
Results - (untitled)

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.



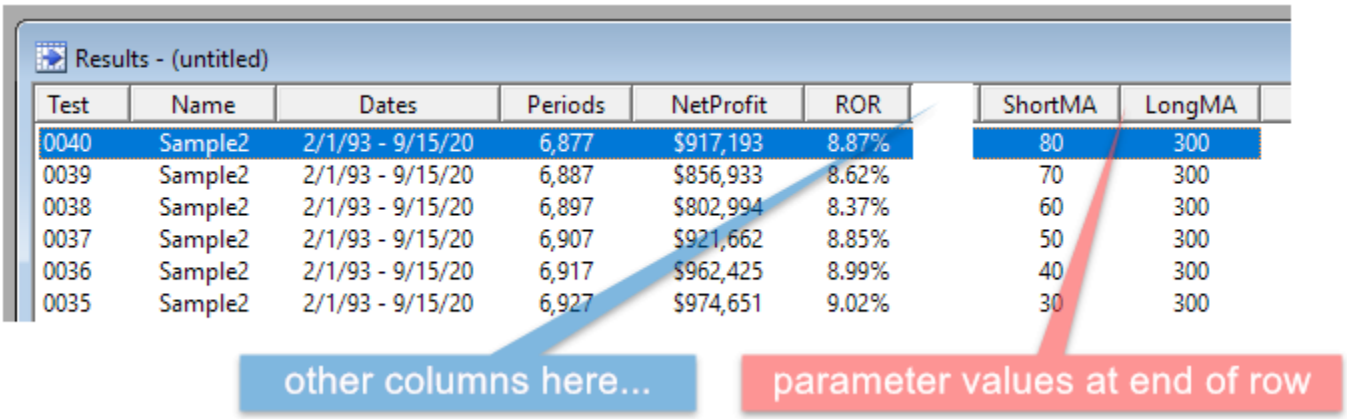
After a graph or chart is opened in this way, it remains linked to the list that it came from. By using the UP and DOWN arrow keys, you can flip through the charts of all trades in a test or symbols in a scan, or the equity curves of all tests in a set of results.

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

6.6.2. 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

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.

The **ResultsByYear** setting (or the toolbar **Outputs** menu) can add a further band of per-calendar-year columns (ROR, MaxDD, and/or Trades) to the right of these.

By default, the results column contents are defined in the script file **Results.rts** located in the same directory as the RealTest program.

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.

6.6.3. 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 List - Test 1 - 14 Trades

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.

Two kinds of rows have no exit: setups that were skipped, kept when **KeepTrades** includes *Skipped* (Side "Skip Long" / "Skip Short", Reason = why the setup was skipped), and positions still open at the end of the test when **EndOfTestExits** is False (Side "Open Long" / "Open Short", Reason "open position", Bars = bars held so far). Both show n/a in the exit columns and are excluded from trade statistics; charting an open-position row shows the entry marker only.

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 List - Test 1 - 5,189 Trades

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn
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	SYY	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
    Days: T.Days
```

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

Trade List - Test 1 - 153 Trades (filtered)

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn
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:

t - Test 1 - 4,075 Trades

Strategy	Sy...	Side	DateIn	TimeIn	QtyIn	PriceIn	Dat...	TimeOut	QtyOut	PriceOut	Reason	Bars	PctGai
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.

6.6.4. 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 Scan 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
```

Scan - 154 Items

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%

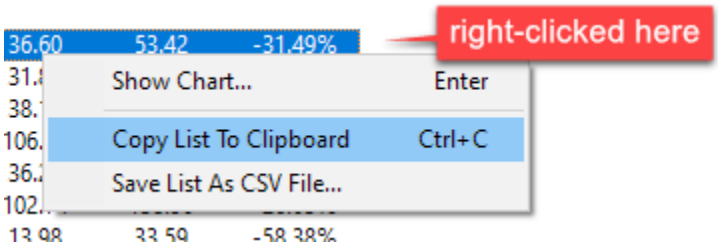
Data can be sorted by any column by clicking on its header. Clicking again reverses the sort order.

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.

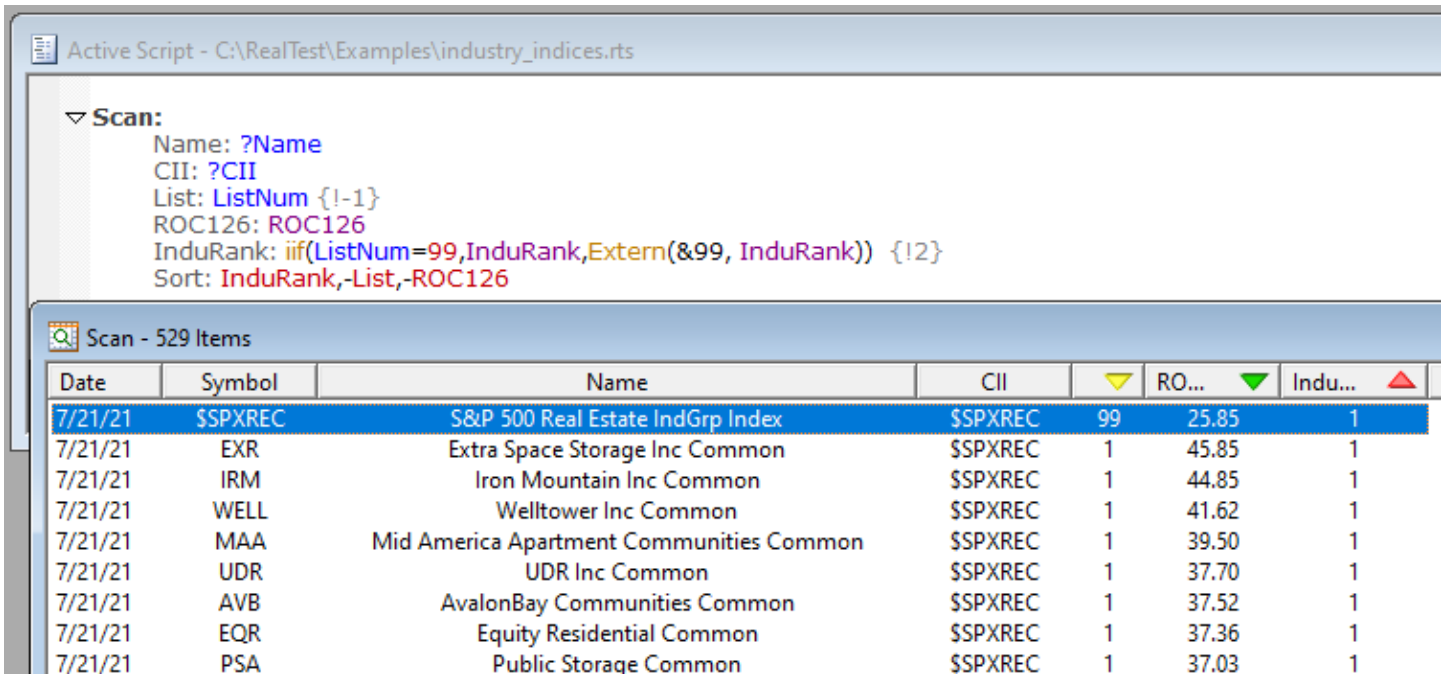
Double-clicking on any row of a Scan Window opens a chart of that stock with the rightmost bar set to the date shown in that row.

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:



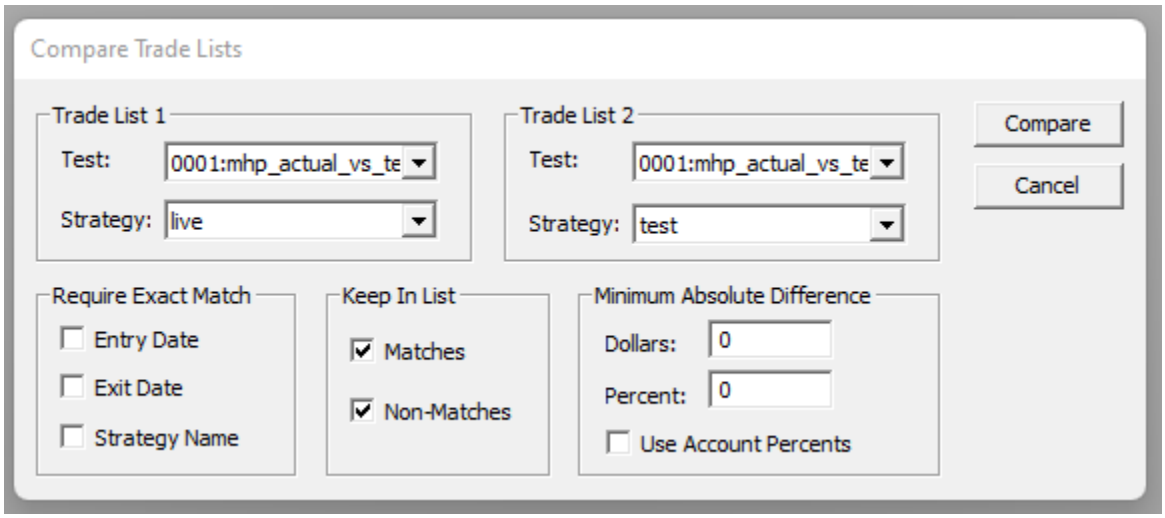
List one or more visible column names to specify the sort priority. Precede a name with a dash to indicate *descending* order, otherwise it will be *ascending*.

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.

6.6.5. 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.



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%
									($898.30)	-2.57%
MOCShortX	1/7/22	479	72.94	1/7/22	479	68.52	$2,112.39	6.05%	$1.22	0.00%
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.

6.6.6. 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.

6.7. 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.

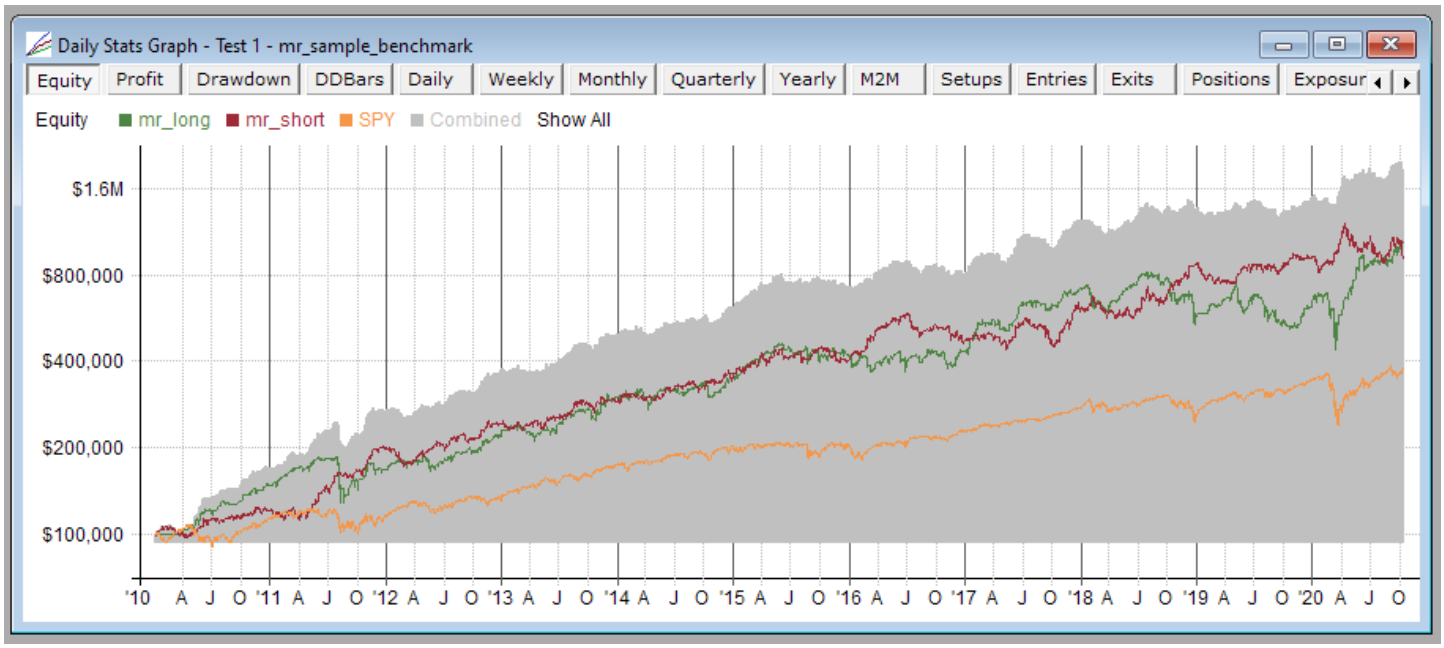
6.7.1. Daily Stats Graphs

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.

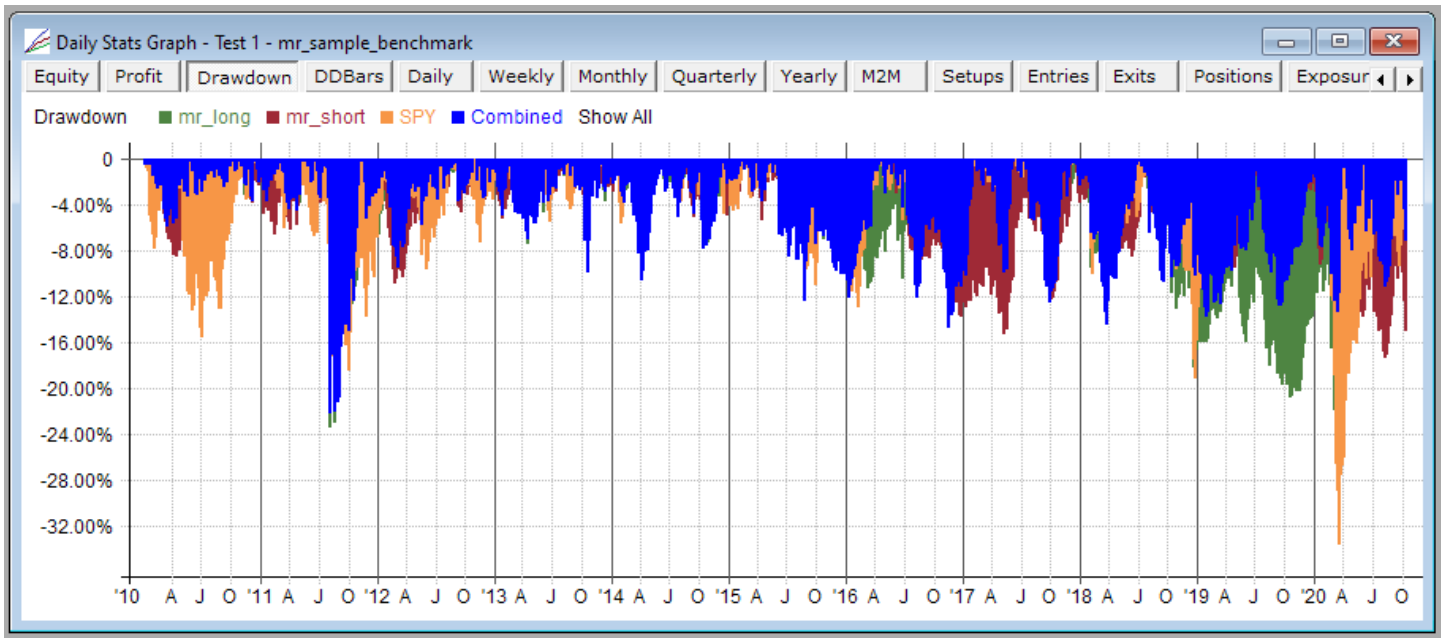
These results time series are also available to the test script while the test is running, allowing a strategy to dynamically refer to its own performance and/or the performance of other simultaneously running strategies. (The example script mr_sample_tracking.rts shows how this is implemented.)

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:

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 shown using the number keys: 1-9 shows only that strategy, and 0 shows only the combined line. Shift+digit adds 10, for systems with more than ten strategies. Ctrl+4 through Ctrl+9 toggle that strategy on or off without affecting the others, and A shows all. (Ctrl+1, 2 and 3 set the pane count instead - see Multiple Panes below.)

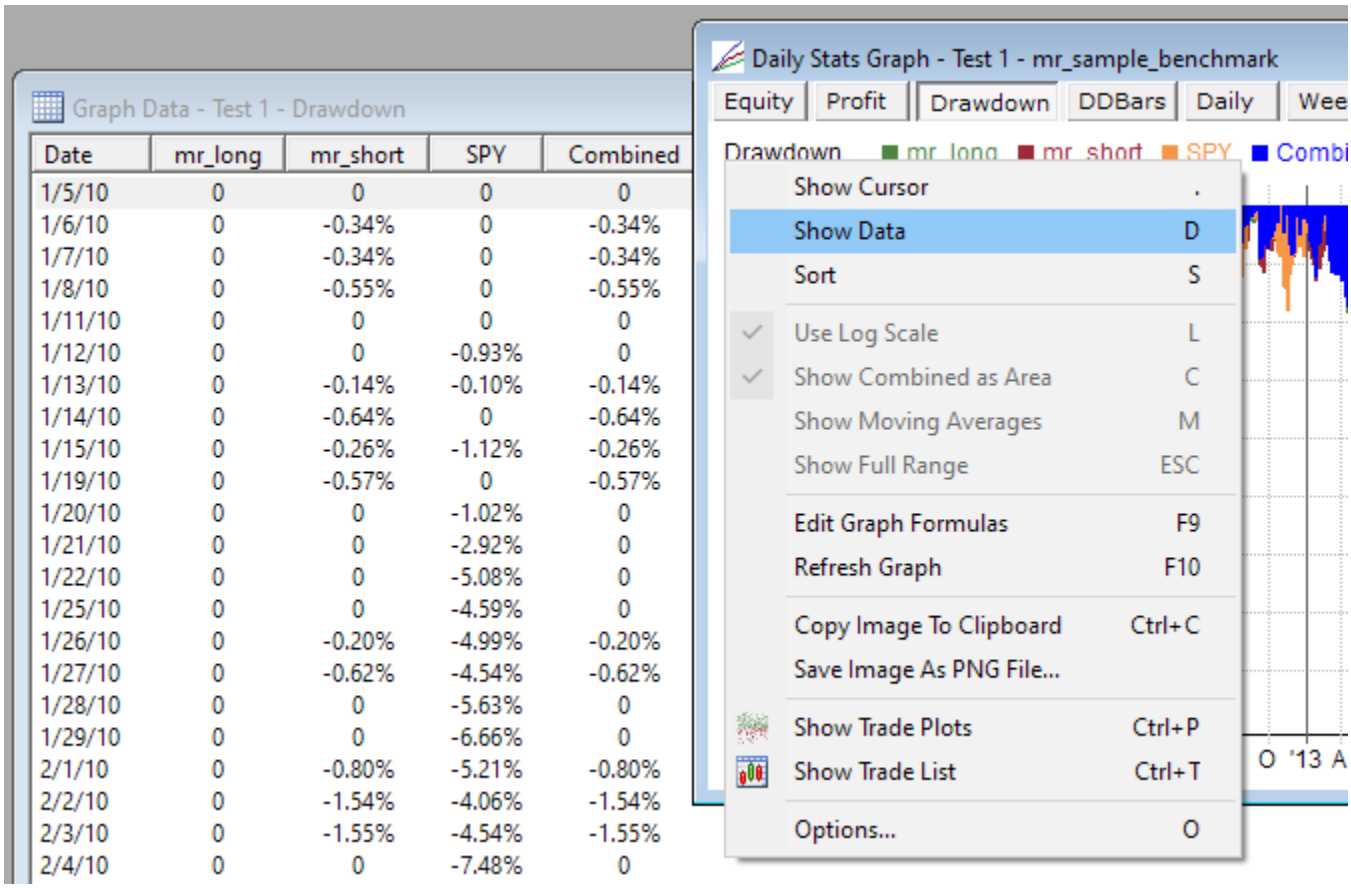
Strategy Side Panel

The strategy list can alternatively be shown as a side panel docked to the left of the graph. Toggle it with the P key, the *Show Strategy Panel* item on the **Graph Menu**, or the "Show as Side Panel" checkbox in the graph options dialog; the choice applies to all graph windows.

When the panel is on and a graph has more than one line, it replaces the legend row at the top of the graph. Each strategy row has a checkbox controlling that line's visibility, and the mouse actions work as they do in the legend: clicking a strategy's name shows only that strategy, Ctrl+click toggles it, and the Show All row shows every line (or hides them all when unchecked).

Drag the divider between the panel and the graph to adjust the panel's width, or double-click it to reset. Copied and saved images always include the legend row rather than the panel.

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

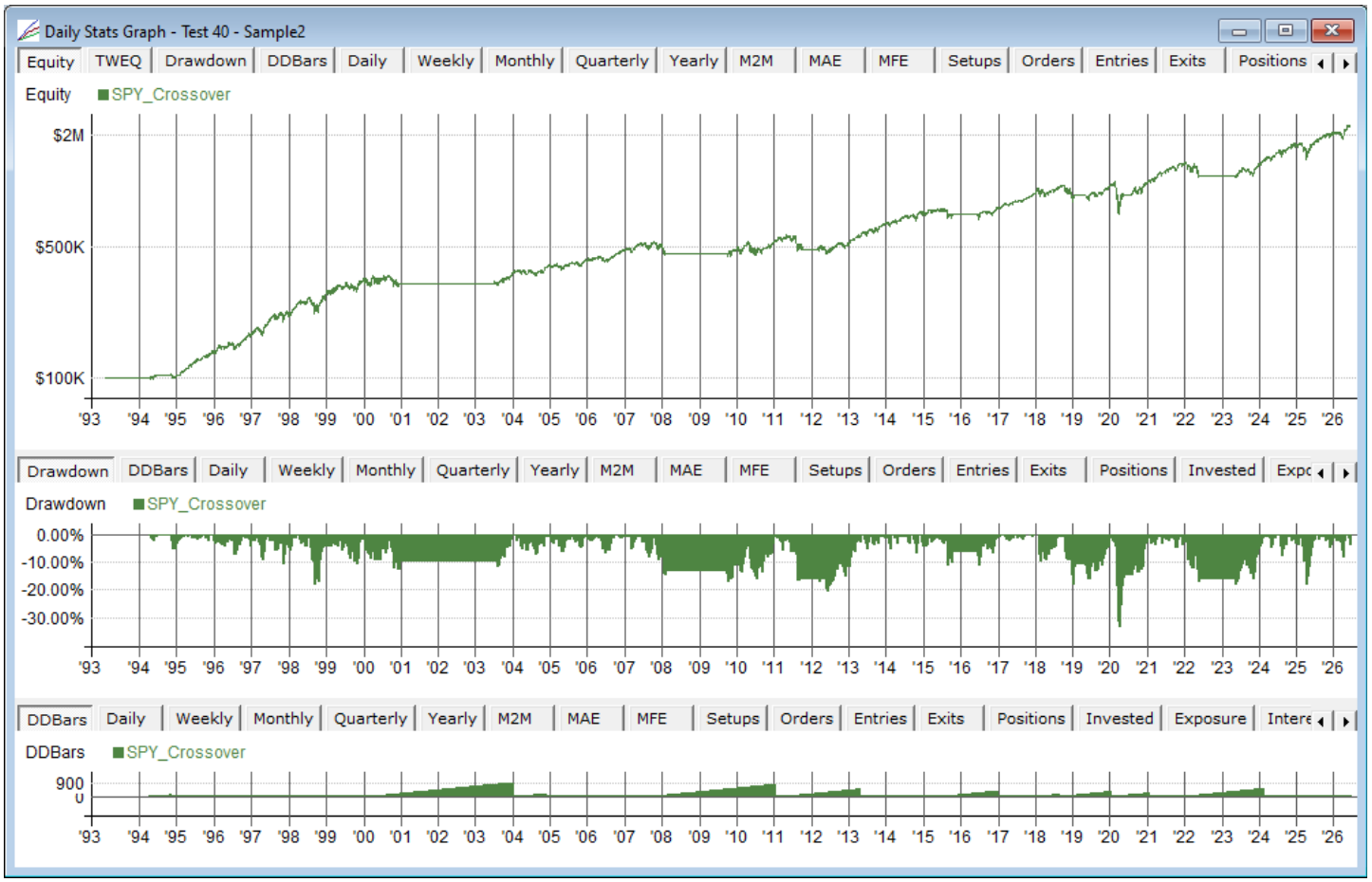


Date	mr_long	mr_short	SPY	Combined
1/5/10	0	0	0	0
1/6/10	0	-0.34%	0	-0.34%
1/7/10	0	-0.34%	0	-0.34%
1/8/10	0	-0.55%	0	-0.55%
1/11/10	0	0	0	0
1/12/10	0	0	-0.93%	0
1/13/10	0	-0.14%	-0.10%	-0.14%
1/14/10	0	-0.64%	0	-0.64%
1/15/10	0	-0.26%	-1.12%	-0.26%
1/19/10	0	-0.57%	0	-0.57%
1/20/10	0	0	-1.02%	0
1/21/10	0	0	-2.92%	0
1/22/10	0	0	-5.08%	0
1/25/10	0	0	-4.59%	0
1/26/10	0	-0.20%	-4.99%	-0.20%
1/27/10	0	-0.62%	-4.54%	-0.62%
1/28/10	0	0	-5.63%	0
1/29/10	0	0	-6.66%	0
2/1/10	0	-0.80%	-5.21%	-0.80%
2/2/10	0	-1.54%	-4.06%	-1.54%
2/3/10	0	-1.55%	-4.54%	-1.55%
2/4/10	0	0	-7.48%	0

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.

Multiple Panes

A graph window can show one, two or three stacked panes, each with its own tab bar and graph selection:



Set the pane count with Ctrl+1, 2 and 3, with the Show 1, 2 or 3 Panes items on the **Graph Menu**, or with the Panes setting at the top of the Options dialog. The second pane initially shows Drawdown and the third DDBars; whatever graph you select in each pane is remembered.

The panes act as one window: the cursor, subset zoom (and ESC), strategy show/hide choices and the display toggles apply to every pane, and the date axes stay aligned. Drag the splitter bar between panes to adjust their relative heights, or double-click it to reset; the proportions are remembered.

Ctrl+C and Save Image capture all visible panes as a single combined image.

Stacked graphs can also appear in Test Summary Reports: a {^2} or {^3} format code in a Graphs section item stacks that graph beneath the preceding {^} graph as one combined report image with a shared date axis. See the **Graphs Section** topic for details.

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:

```
Active Script - C:\RealTest\Graphs.rts
Notes:   this is the default set of stats to offer in Graph windows
         edit this file then press F4 or click Apply to change your set of available graphs

         graphs with | in the format code are shown as bars rather than a line, and can be sorted
         graphs with ^ in the format code are included in summary reports

         you can also add a Graphs section to any script, to override these
         this default one is only used when there is not a custom one

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
```

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

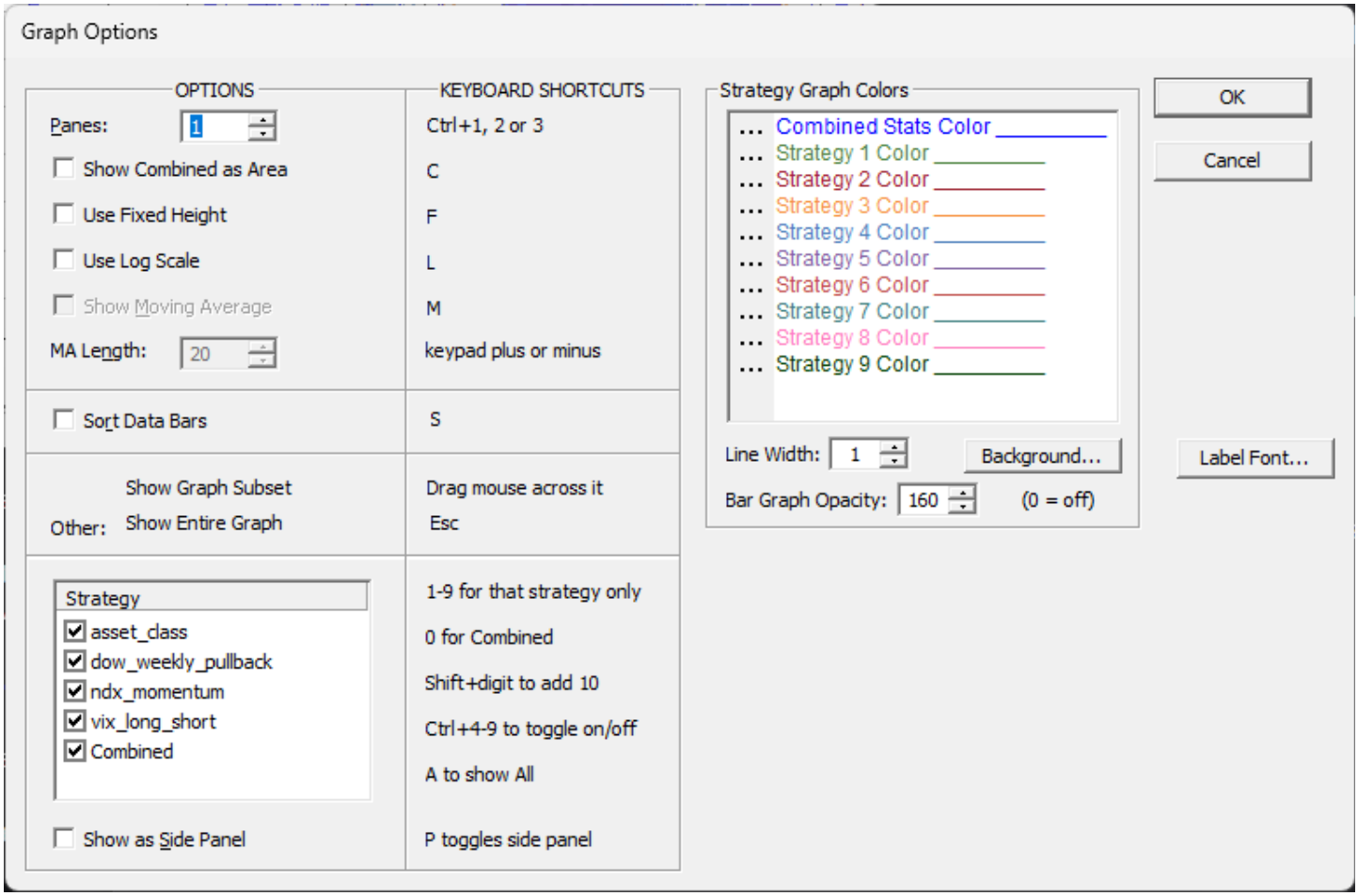
Graph windows allow specification of 10 colors via the options dialog.

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.

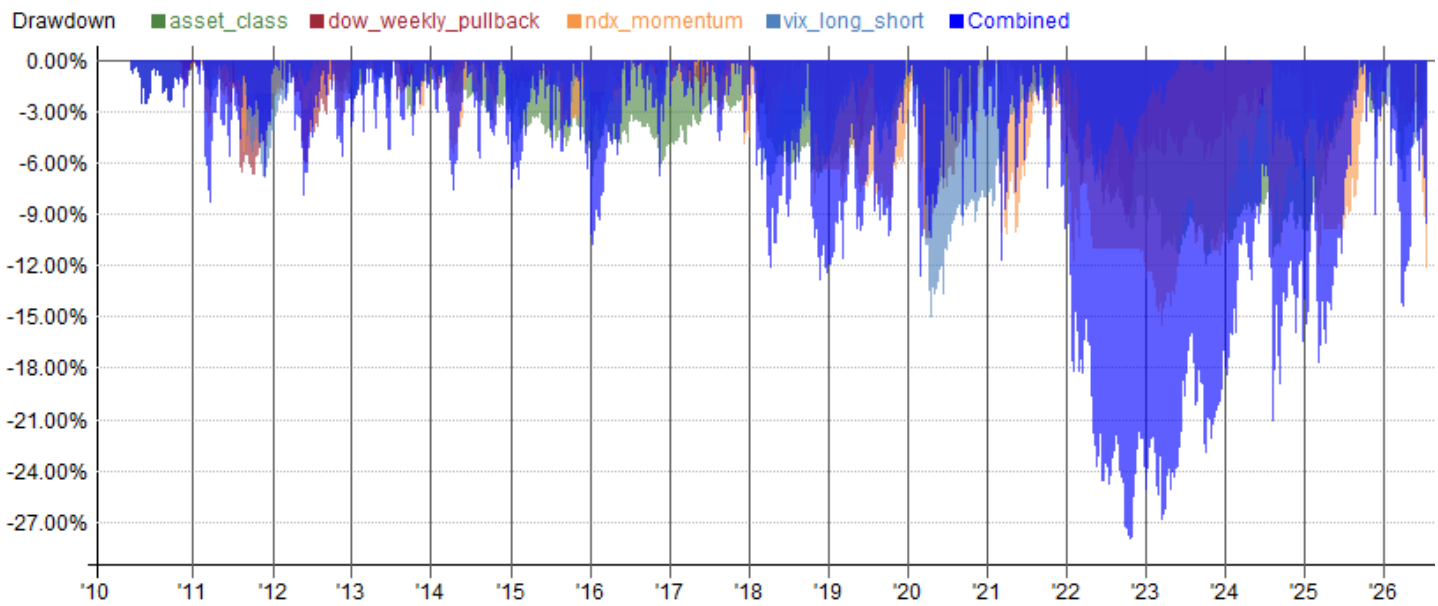
Line colors can be edited using the graph options dialog:



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

Bar Graph Opacity makes the fills of bar-style graphs (such as *Drawdown*) partially transparent, so that overlapping bars from multiple strategies remain visible through each other and the grid lines show through. The value is an opacity from 1 (nearly invisible) to 255 (fully opaque); 0, the default, turns the feature off and draws bars fully opaque as before. A strategy or chart color that specifies its own alpha channel (see **Output Formats**) overrides this setting for that item.

Here is the drawdown graph of a five-strategy system with Bar Graph Opacity set to 160:



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.

6.7.2. Optimization Results Graphs

An Optimization Graph window appears when you select *Show Optimization Graph* from the **Results Menu** or press the

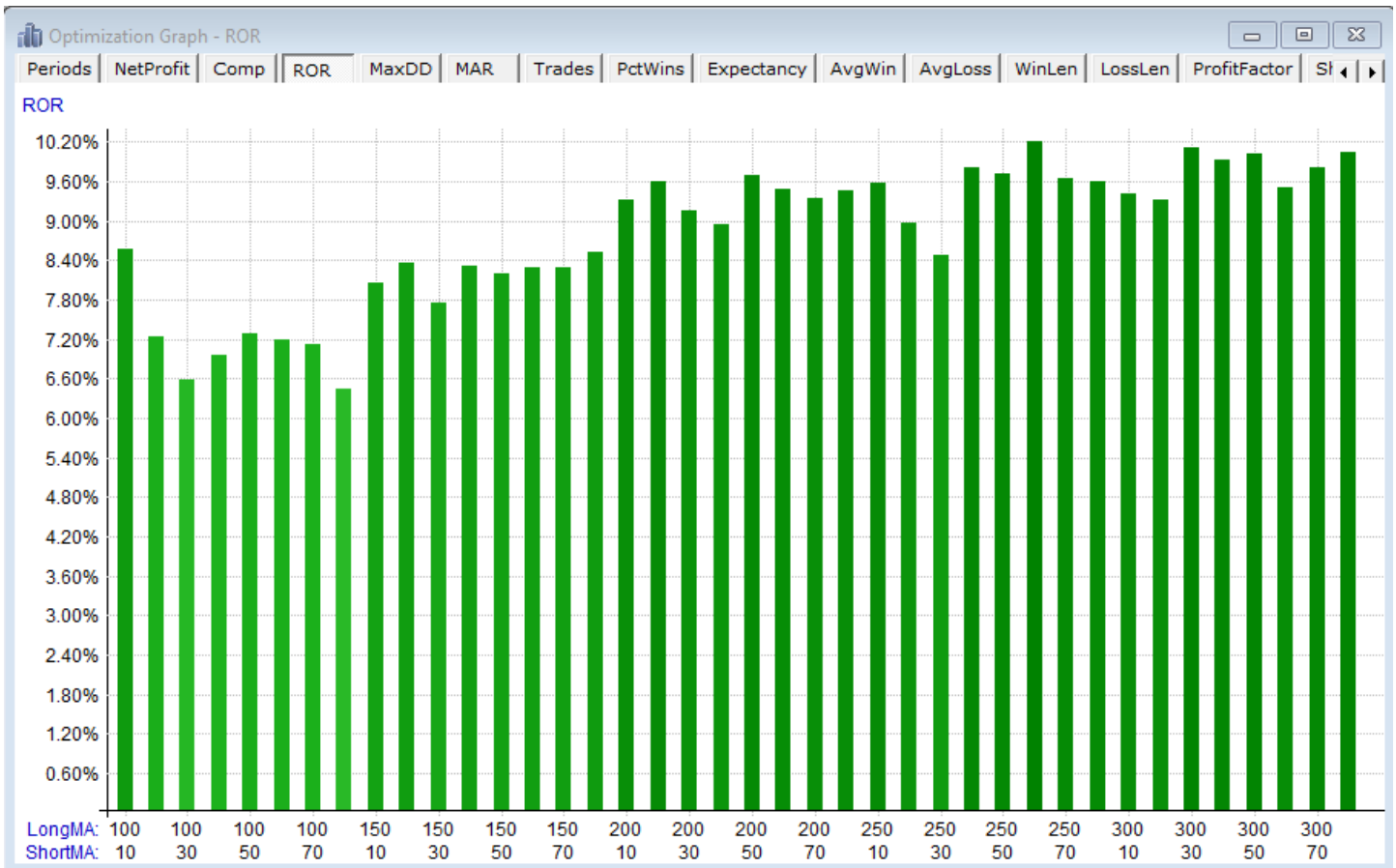
button on the Tool Bar.

It graphs the values of one Results column across every test in the Results window. The tabs across the top of the window (or the Left and Right arrow keys) switch which column is being graphed.

Seven graph types are available. Select one from the ResGraph menu, from the Options dialog, or directly by pressing its shortcut key:

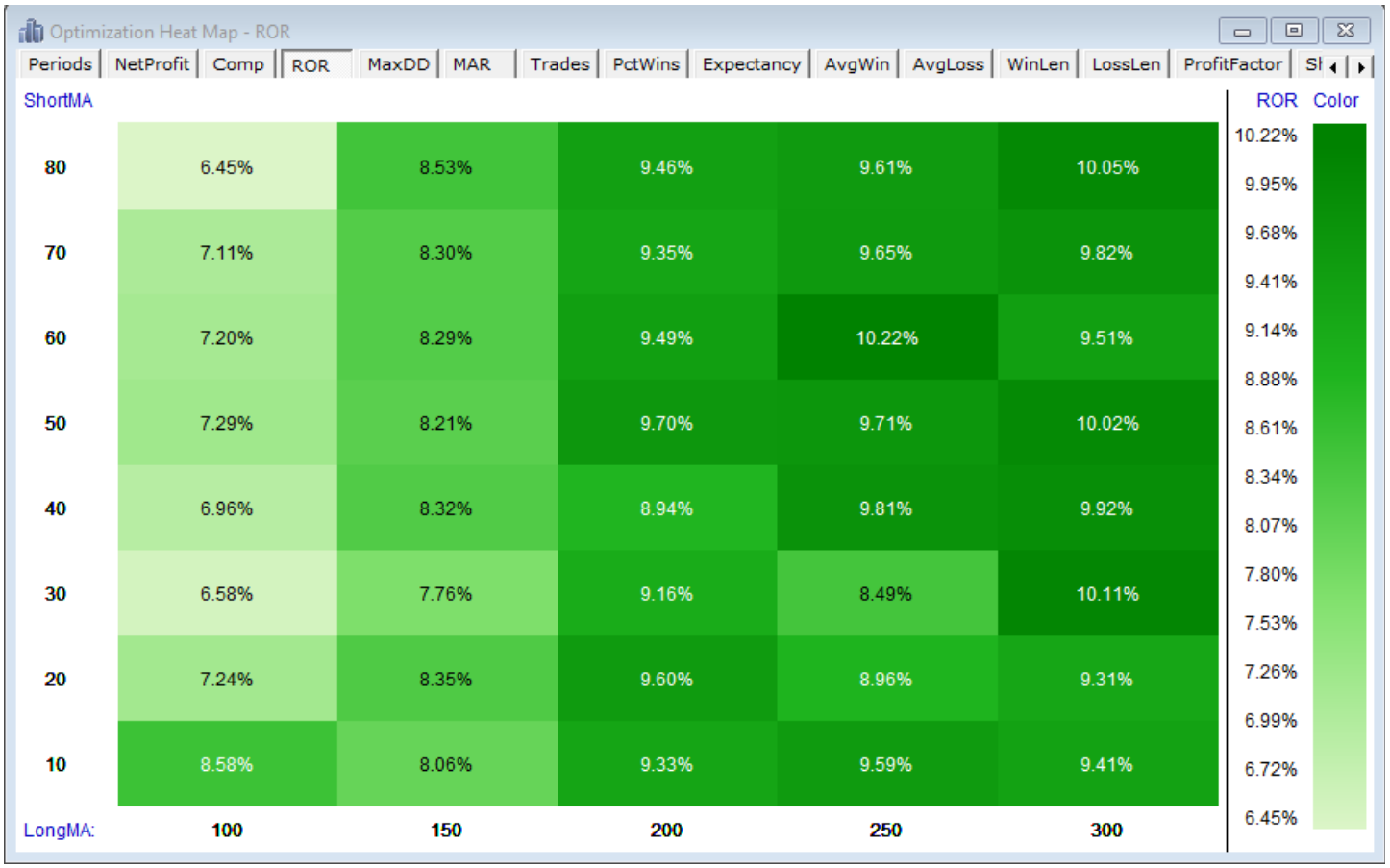
- **Bar Chart (B)** - one bar per test, labeled with the parameter values that produced it
- **Heat Map (H)** - a grid of cells for two parameters at once, colored by value
- **3D Surface (3)** - the heat map grid drawn as a rotatable three-dimensional surface
- **Histogram (M)** - the distribution of the column's values across all tests, with the selected test highlighted as the baseline
- **Equity Curves (E)** - the combined equity curve of every test, overlaid on one date axis
- **Drawdown Curves (D)** - the combined drawdown curve of every test, overlaid on one date axis
- **Overfitting Analysis (F)** - the most recent optimization run with **OptOverfit** enabled, in two tab views: Selection (CSCV/PBO) and Surface (Walk-Forward Correlation); F again switches between them - see **Overfitting Analysis**

Bar Chart



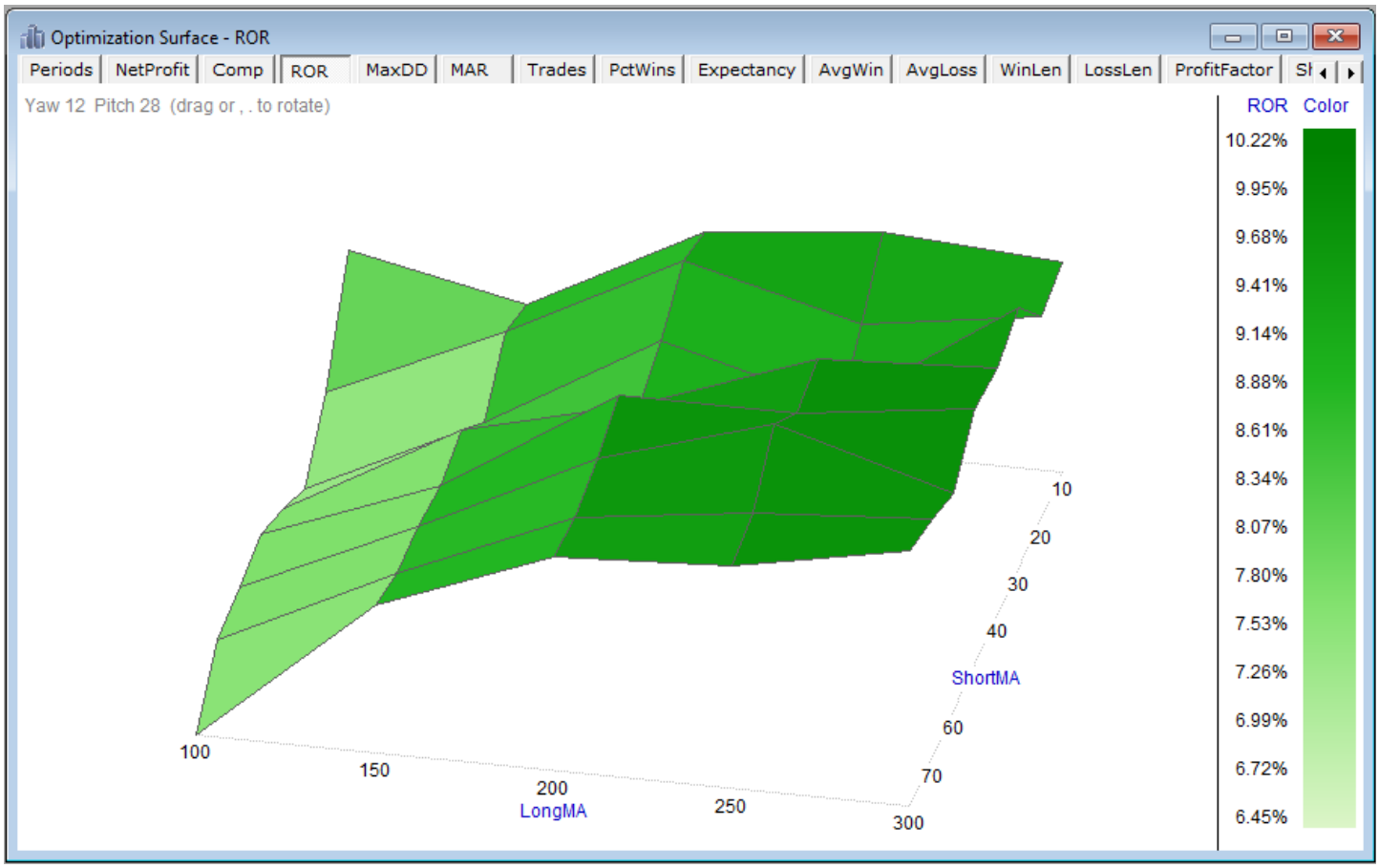
Each bar is one test, with the value of every varied parameter shown beneath it. Hover over a bar to see its exact parameter values and result. *Sort Columns* (S) orders the bars by value rather than by parameter, which makes the range of outcomes easier to judge. *Linear Regression Line* (L) overlays a trend line.

Heat Map



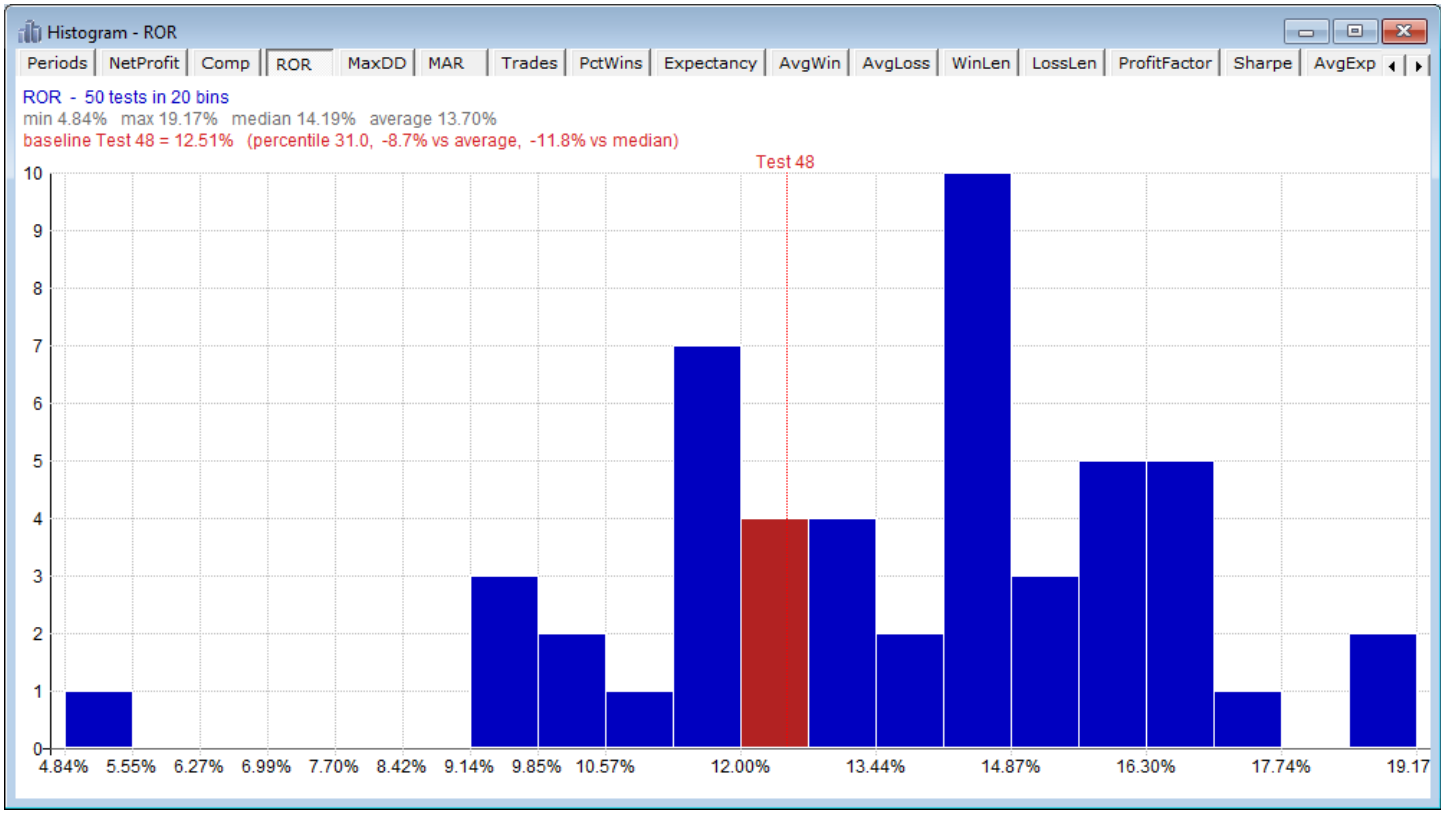
When two or more parameters were varied, the heat map shows one parameter per axis with each cell colored according to the value scale on the right. Choose which parameters to map using the X-Axis and Y-Axis submenus of the ResGraph menu, or the Options dialog. Colors run from pale near zero to deep green (positive values) or deep red (negative values) far from zero. Hover over any cell for its exact values, or press V (**Show Values**) to draw every cell's value in the cell; the values are drawn only when the cells are wide enough to hold them. **Show Data** on the ResGraph menu opens the same grid as a table (one row per Y-axis value, one column per X-axis value) that can be copied into a spreadsheet.

3D Surface



The same grid as the heat map, drawn as a surface. Drag with the mouse to rotate it freely, or press the , and . keys to rotate in 15-degree steps. The viewing angle is remembered between sessions. A robust parameter neighborhood appears as a broad plateau; a lone spike suggests an over-fit combination.

Histogram



The histogram divides the value range of the current column into bins and counts how many tests land in each - the distribution of outcomes across the whole optimization. It is made for parameter

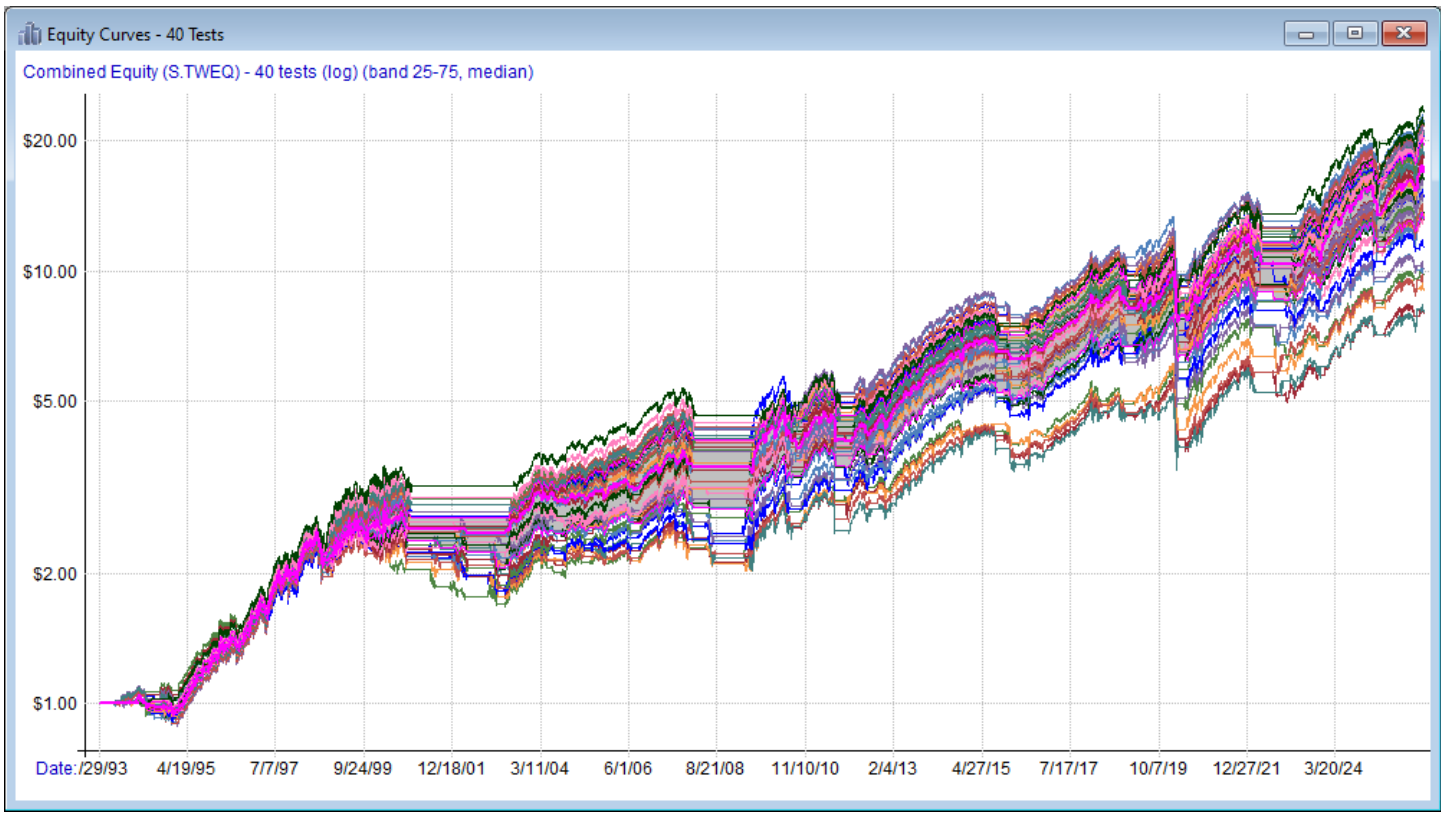
sensitivity analysis: run an optimization that varies your parameters around their chosen values, then see where those chosen values fall in the range of possible outcomes. Hover over any bar for its exact value range and test count.

One test is the **baseline** - normally the test that used your chosen parameter values. Its bin is drawn in red with a dotted marker line, and the header shows its value, its percentile rank, and its difference from the average and median of all tests. The baseline is simply the selected row of the Results window: click a different row, or press the Up and Down arrow keys on the graph, to move it. A baseline near the middle of the distribution suggests the outcome is typical of the whole parameter neighborhood; one in the top few percent may owe more to a lucky parameter draw than to the strategy - see **Overfitting Analysis** for a complementary test of the same question.

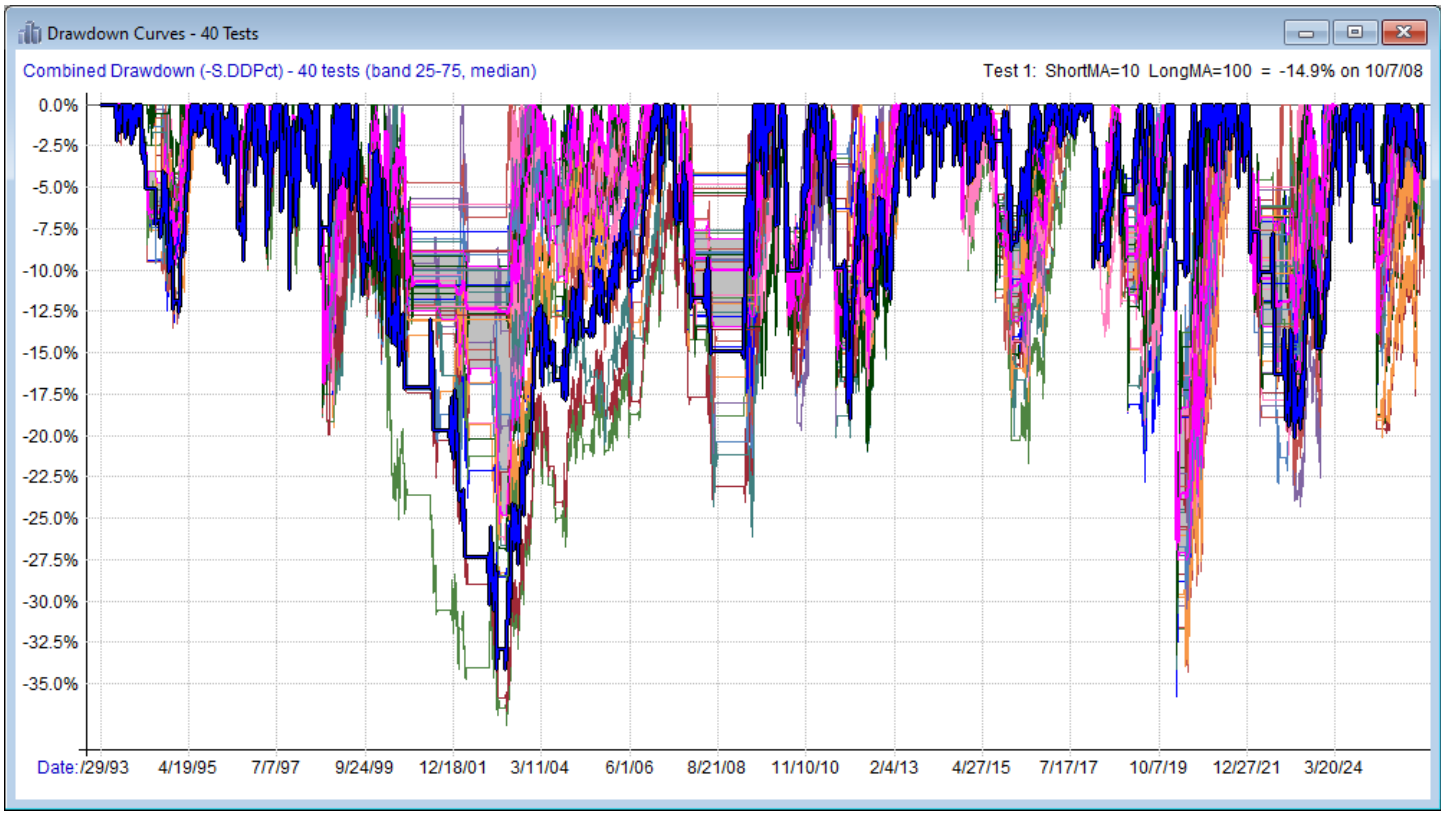
The number of bins is chosen automatically from the test count; the *Bins* option in the Options dialog sets a fixed count instead (0 = automatic).

Log Summary Text on the ResGraph menu writes a tab-delimited table to the Log window with one row per results column: the minimum, maximum, median, average and standard deviation across all tests, plus the baseline's value, percentile rank, and differences. Paste it into a spreadsheet for further analysis.

Equity and Drawdown Curves



Equity Curves draws the combined equity of every test in the list (the **S.TWEQ** daily series) as its own line on a merged date axis - a direct view of how sensitive the strategy's outcome is to parameter choice. *Log Scale* (G) switches the equity axis to logarithmic. *Percentile Band* (P) shades the 25th-75th percentile range and draws the median path in magenta, as shown above.



Drawdown Curves is the same view of each test's combined drawdown (**S.DDPct**).

Hovering over either curve graph highlights the nearest curve and shows its test number, parameter values and final value; clicking selects that test's row in the Results window, so you can pull any interesting line out of the bundle for closer study.

The curve graphs need the per-period stats of each test, so they are not available for results that were saved with *Discard Stat Details* (**OptNoStats**). Press ESC to cancel drawing the curves of a very large results list.

Averaging Method

When more than one test shares the same X-axis (and Y-axis) parameter values - for example when a third parameter was also varied, or Test Iterations was greater than one - each bar, cell or surface point aggregates those tests. The *Averaging Method* option chooses how: Mean (the default), Median, Minimum or Maximum. Hover readouts indicate how many tests went into the value, e.g. "(avg of 5)".

Options and Images

Press O (or choose *Options* from the ResGraph menu) to open the Optimization Graph Options dialog, which collects every choice in one place. The options are grouped by the graph type they apply to, and the group captions for the currently selected type are shown in bold; the Dark Background option applies to all types. All settings persist between sessions.

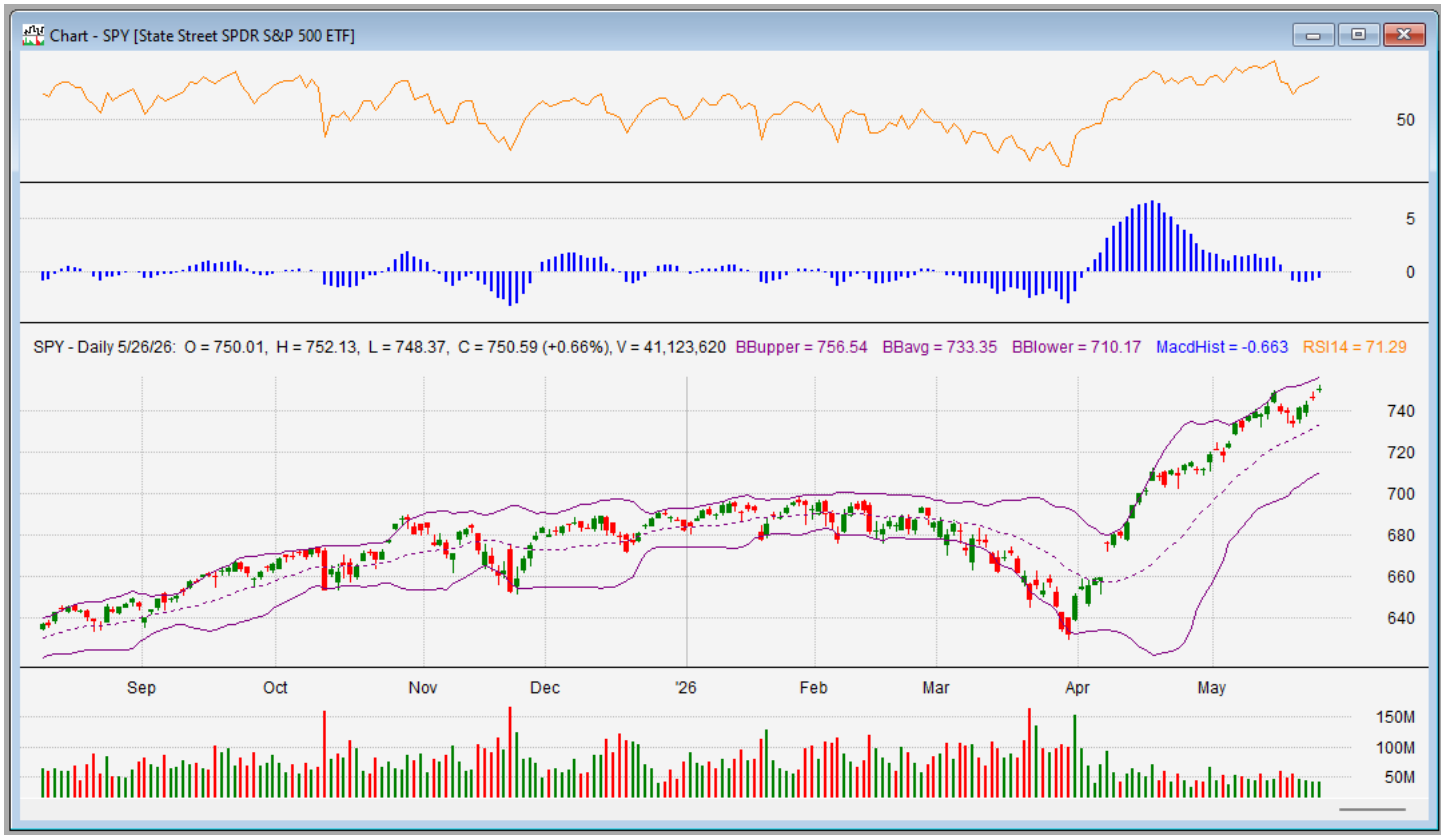
Ctrl+C copies the current graph to the clipboard as an image, and *Save Image as PNG File* on the ResGraph menu saves it to disk.

If you haven't already done so, please go through **Tutorial 2** for a worked optimization example.

6.7.3. 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 can include up to five panes: a price pane in the middle, up to three optional upper indicator panes stacked above it, and an optional lower pane below it. Each pane automatically appears when it has content and disappears when it does not.

The price 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 upper indicator panes – named *ind1* (closest to price), *ind2*, and *ind3* (on top) – each show any number of indicators sharing the same scale. An indicator is routed to a specific pane by including a *{pane: ind1}* (etc.) attribute in its formula, or by using the shorthand *{^}* (= *^1*), *{^2}*, *{^3}*. See **Output Format Specification** for the full attribute syntax (including per-line *color* and *line* styles: *Line*, *Bars*, *Dashed*, *Dotted* – and per-line *width*).

The lower pane shows color-coded volume bars, or can show another group of custom indicators (sharing the same scale) routed there with *{|}* or *{pane: volume}*. It appears automatically when it has content or when the data includes volume bars, and disappears otherwise.

The horizontal dividing line between any two consecutive active panes can be dragged up or down with the mouse to adjust their relative heights independently.

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. The N key toggles the grid lines
6. The T key cycles the trade labels option (None, Selected, All) on a trade chart
7. 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

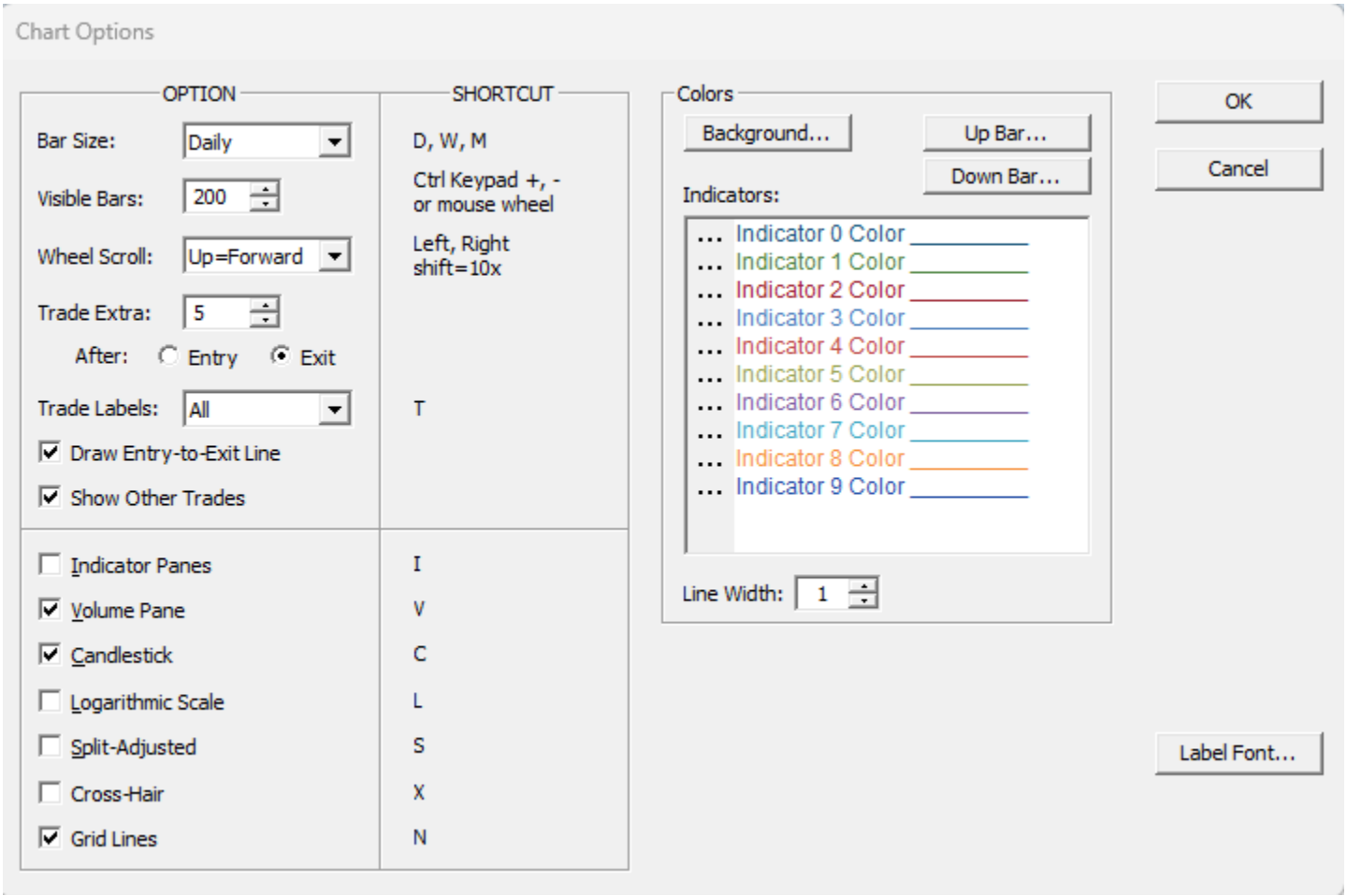
6. Zoom in/out by doing any of the following:
   - enter a visible bar count in the options dialog
   - hold down Ctrl and roll the mouse wheel
   - press the + or – keys
   - grab and drag either end of the scrollbar thumb
   - if the caret is not visible, click and drag across a subset of the visible 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)

         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        formula          {format}
// Bollinger bands on chart
         BBupper:    BBTop(20, 2)     {color: purple}
         BBavg:      MA(C, 20)        {color: purple, line: dotted}
         BBlower:    BBBot(20, 2)     {color: purple}

// MACD Histogram in indicator pane 1
         MacdHist:   MACDH(12,26,9)   {pane: ind1, line: Bars, color: blue}

// RSI in indicator pane 2
         RSI14:      RSI(14)          {pane: ind2, color: orange}

// 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. Most of them can also be set from a script - see **Chart Settings**.



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.

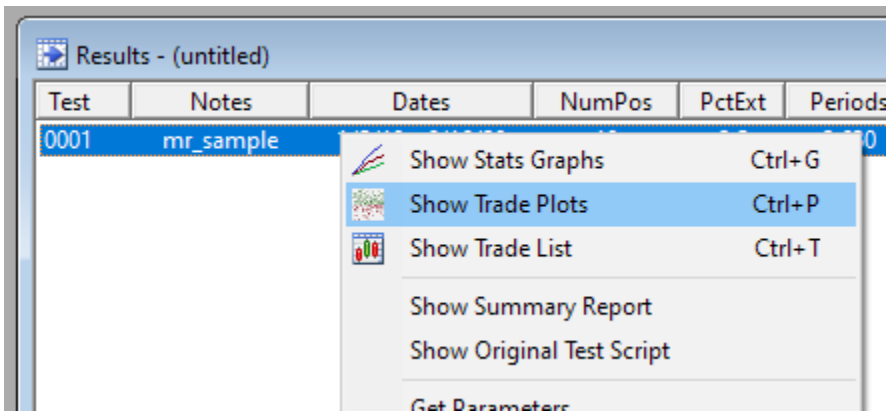
Trades are marked with arrows drawn just outside the bar's high-low range, in the strategy's color. The arrow points the way the order went (up = bought, down = sold) and its fill tells entry from exit: a filled arrow opens a position and a hollow arrow closes one, so a long trade is a filled up-arrow below its entry bar and a hollow down-arrow above its exit bar, and a short trade the reverse. A skipped entry is a dotted hollow arrow, and a position still open at the end of the test has an entry arrow only. The chart's own trade also gets a dot at each fill price, the optional entry-to-exit line, and (unless Show Other Trades is on) a detail row below the legend.

The "Trade Labels" setting controls which markers get text labels (buy/short at the entry, the exit reason at the exit): None, Selected (only the chart's own trade) or All (the default). Labels are placed on the outer side of the arrow and are dropped rather than drawn on top of one another when trades are dense. When several trades enter or exit on the same bar (as a dynamically sized strategy does when it rebalances), they share one arrow whose label carries the count, e.g. "buy x3". Hovering the mouse over any marker shows a tooltip with the trade's details, or one line per trade for a shared marker, and clicking the marker of another trade selects that trade (in the trade list as well), re-framing the chart on it; each click on a shared marker moves on to the next of its trades.

The indicator color list specifies the default colors of indicator lines that don't specify a color in their formulas.

6.7.4. Trade Plots and Analysis

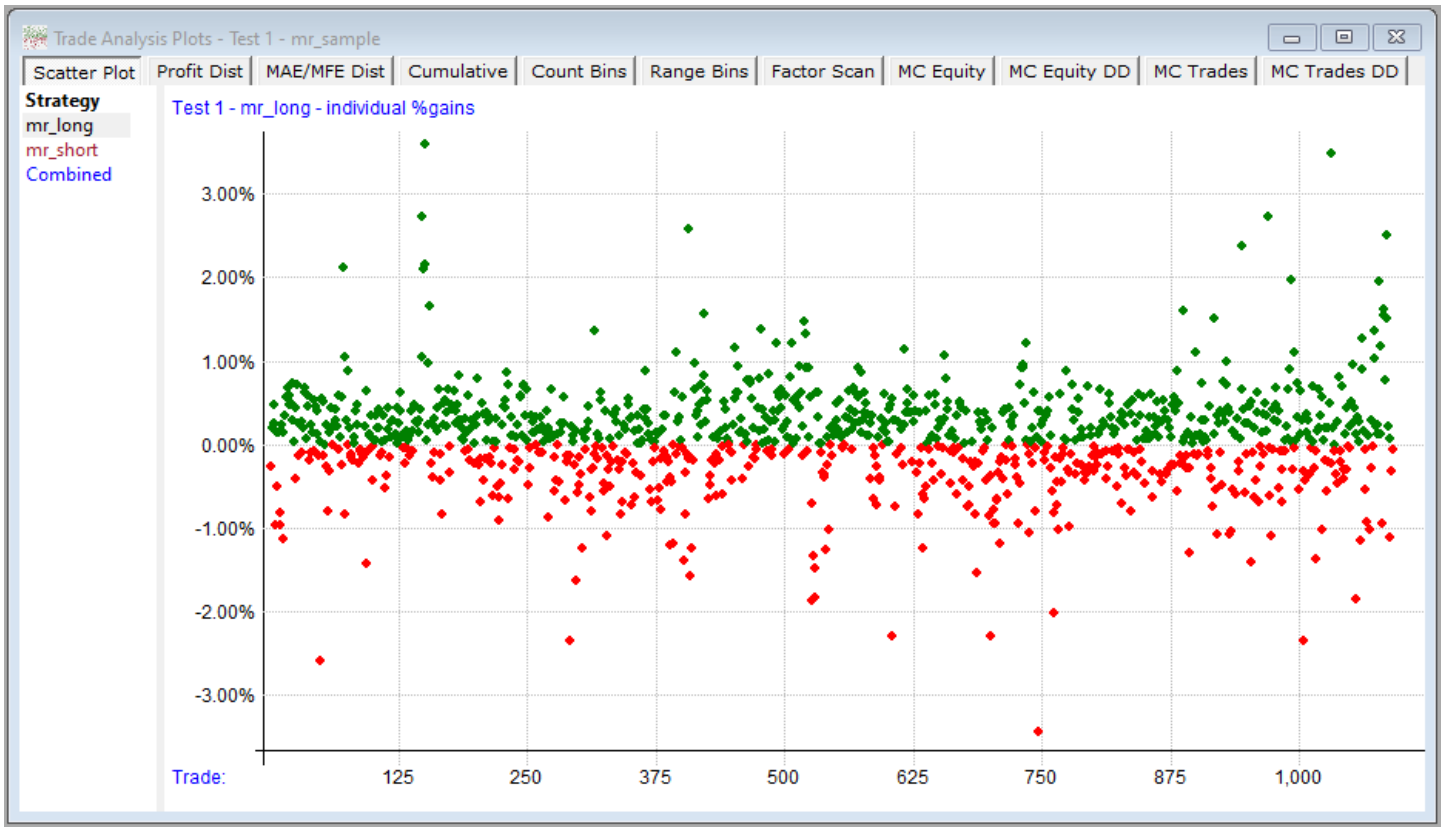
To view a trade plot and access all the trade analysis features, right-click on a test in a **Results Window** and select **Show Trade Plots**.



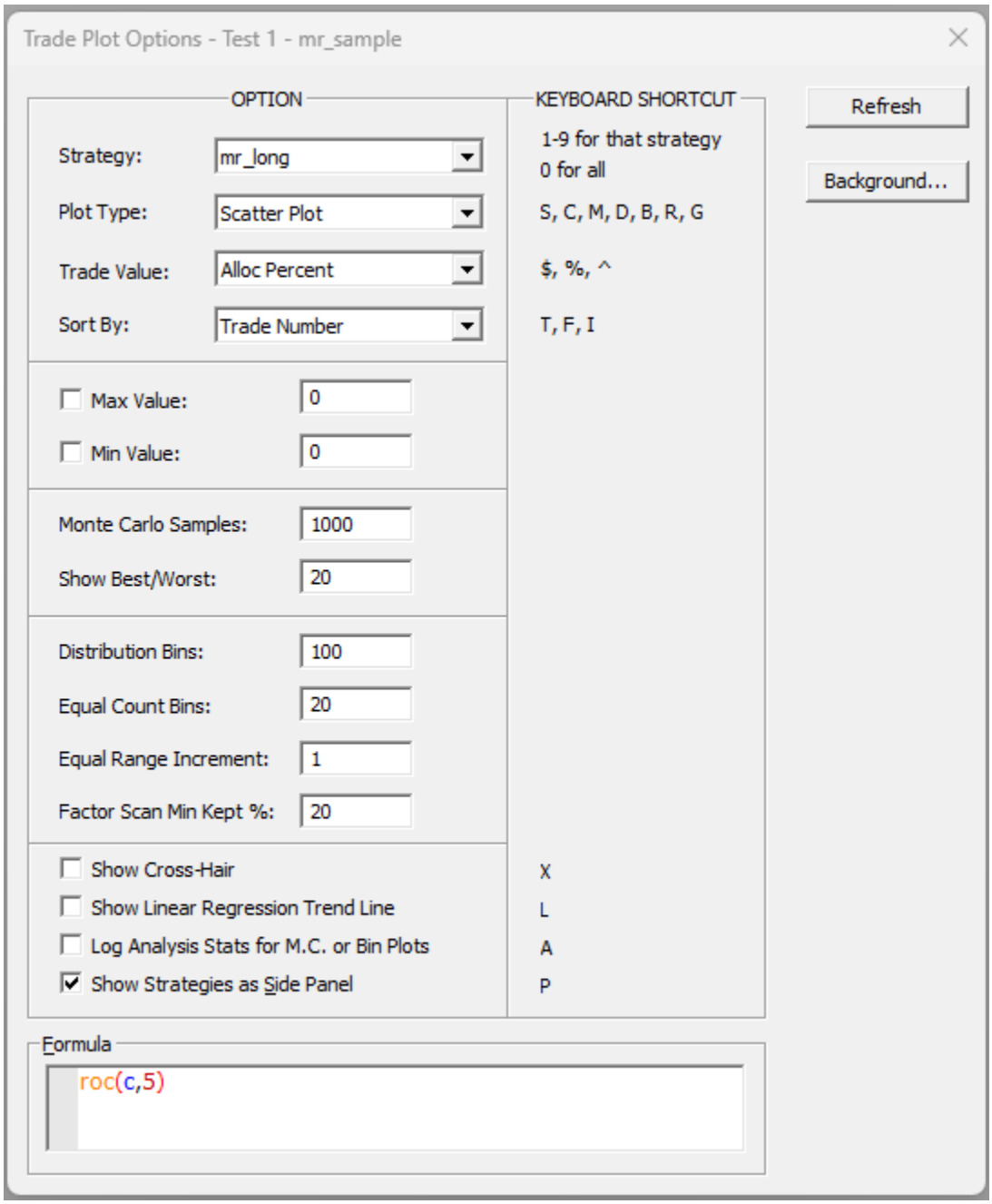
The type of plot shown will be whichever type you last viewed.

A tab bar across the top of every plot window shows the available plot types; clicking a tab is the quickest way to switch the display. (The Plot Type list in the options dialog and the keyboard shortcuts do the same thing.) The tabs and the Plot Type list use abbreviated names - Profit Dist, MAE/MFE Dist, Cumulative, Count Bins, Range Bins, Factor Scan, MC Equity, MC Equity DD, MC Trades, MC Trades DD - for the plot types described by their full names below.

When the test contains more than one strategy, a strategy list is docked to the left of the plot. Exactly one strategy is always selected; clicking a name (or pressing that strategy's number key, 1-9) switches the plot to that strategy's trades. Toggle the panel with the P key, the *Show Strategy Panel* item on the Plot menu, or the "Show Strategies as Side Panel" checkbox in the options dialog, and drag the divider between the panel and the plot to adjust its width. Copied and saved images exclude the tab bar and the panel.



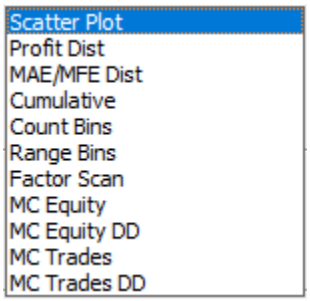
Once a plot window is open, right-click and select Options (or press the O key):



This dialog serves as the control panel for the plot window. It stays open while you use the plot window, making it easy to quickly try the different plot types and options.

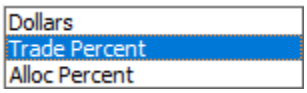
Strategy selects the strategy or statsgroup for which to plot the trades (equivalent to the side panel described above).

Plot Type selects the type of plot to display (the same choices as the tab bar):



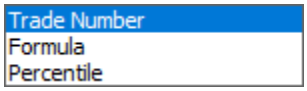
Each plot type is described below.

Trade Value selects the units to display on the Y-axis:



Choices are: dollar profit, percent gain on position size, or percent gain on equity allocation.

Sort By specifies the units of the X-axis:



Trade Number plots the trades in the order they were entered. **Formula** sorts the trades by the value of the Formula field (described below), with the X axis scaled by that value. **Percentile** uses the same formula sort, but places each trade at its rank percentile (0 to 100) rather than at its formula value, spreading the trades evenly along the X axis. This is useful when a few outliers would otherwise compress most of the plot into a narrow band. In the Equal Range Bins plot type, the bins themselves remain value-based and only the bin boundary labels are converted to percentiles. In the **Factor Scan** plot type, Trade Number shows the selected factor's kept and discarded trades over time, and Formula and Percentile show its cutoff profile.

Min Value and **Max Value** make it possible to filter outliers that might distort the plot.

Monte Carlo Samples and **Show Best/Worst** are options for the Monte Carlo plots which are described below.

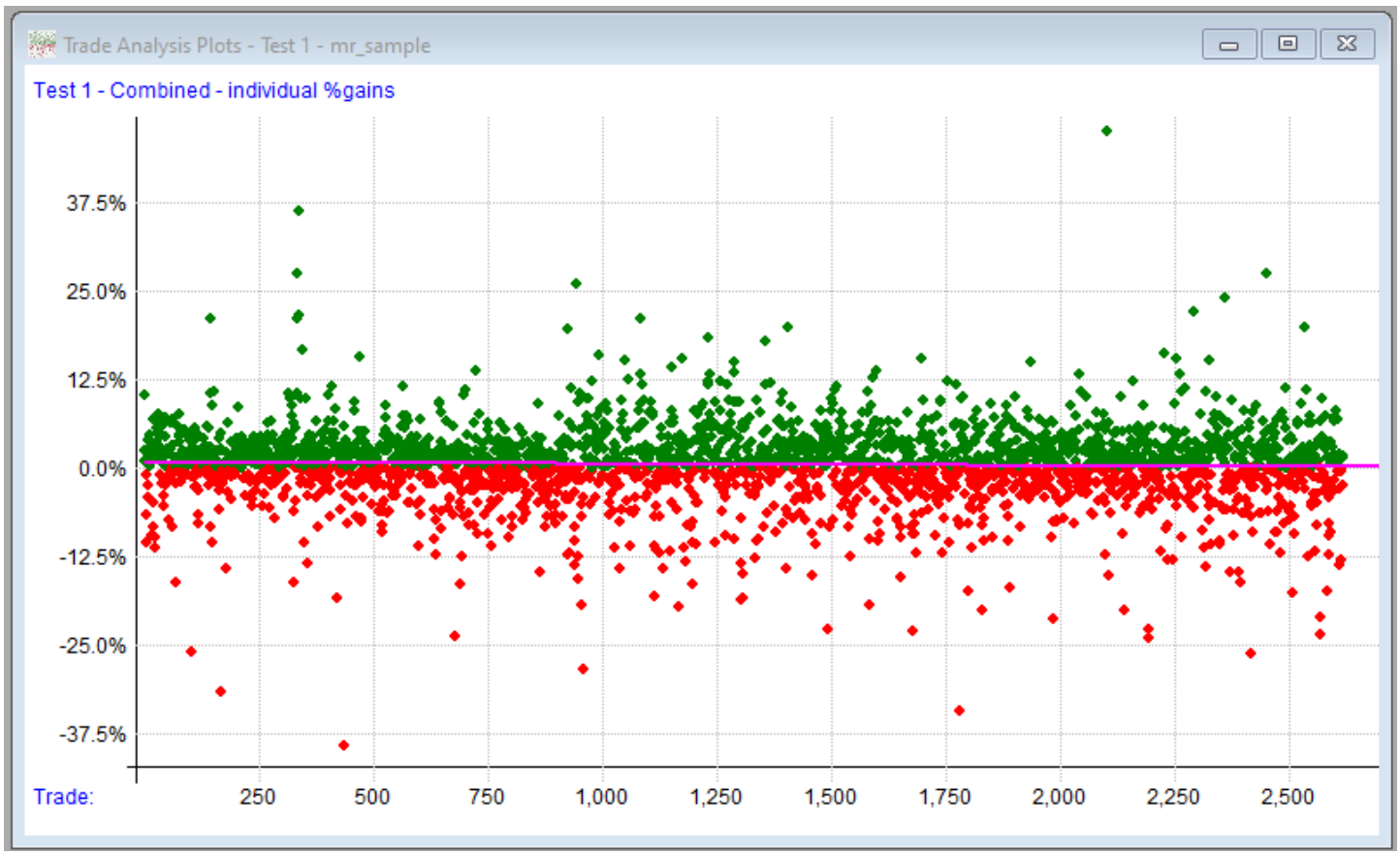
Distribution Bins, **Equal Count Bins** and **Equal Range Increment** are options for those plot types, also described below. **Factor Scan Min Kept %** is the retention floor for the **Factor Scan** plot type.

Show Cross-Hair, **Show Linear Regression Trend Line**, and **Log Analysis Stats** are additional display and output options.

Notice that some of these items have keyboard shortcuts shown next to them. Pressing that key when the plot window is active is equivalent to selecting the item in the options dialog.

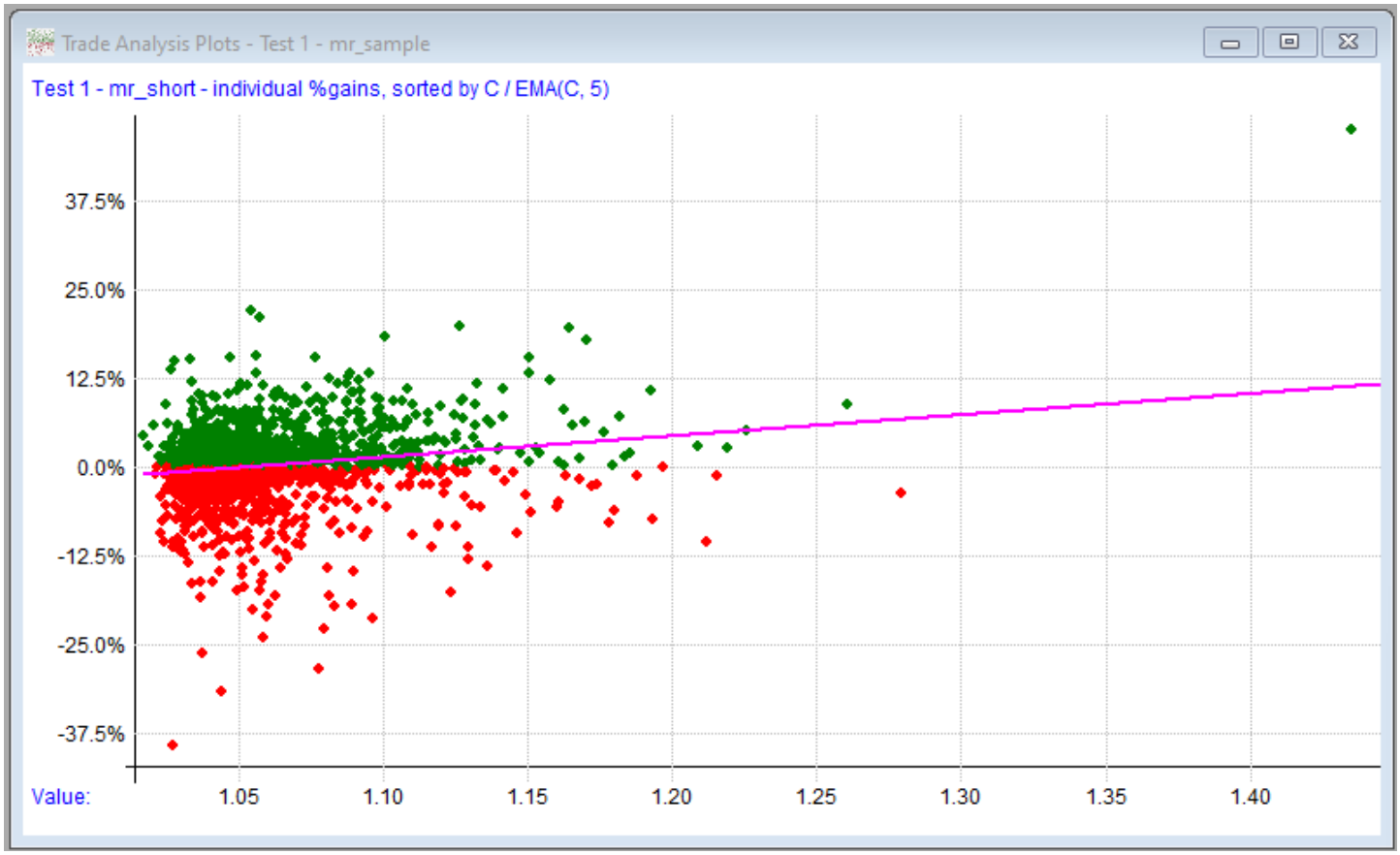
Formula is a miniature script editor where you can type any formula expression from the **RealTest Script Language**. The context of evaluation for the formula is the "setup bar" of each trade, i.e. the **latest available bar** at the time it was decided to enter the position. This makes it possible to quickly sort a trade plot by a variety of "entry factors" to visually assess whether each factor has "an edge". This will be demonstrated below.

The simplest kind of trade plot is the **Scatter Plot**:



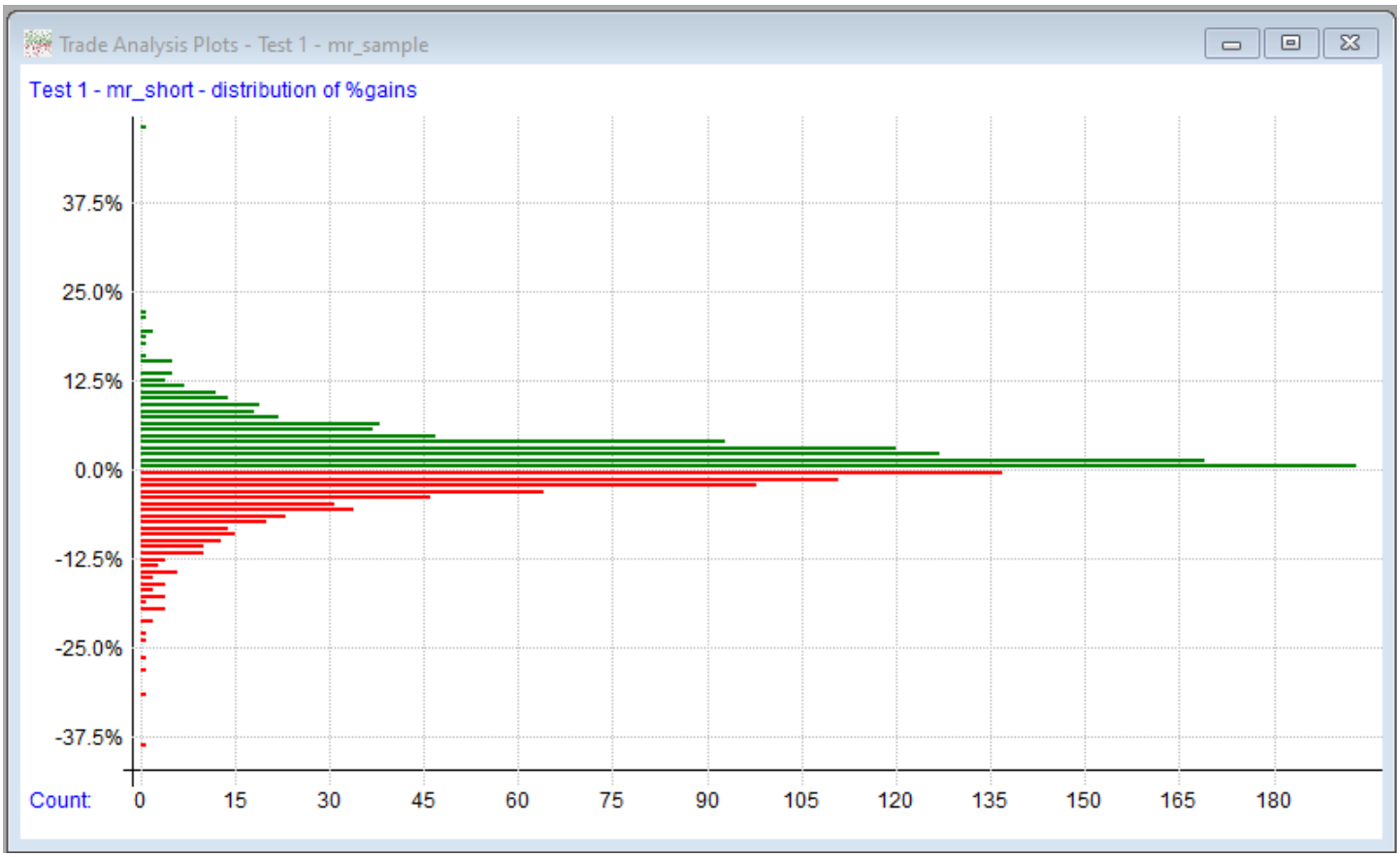
This gives a feel for the general distribution of trade results.

Here we've changed the plot to the mr_short strategy and sorted the results by the **C / EMA(C, 5)** formula:

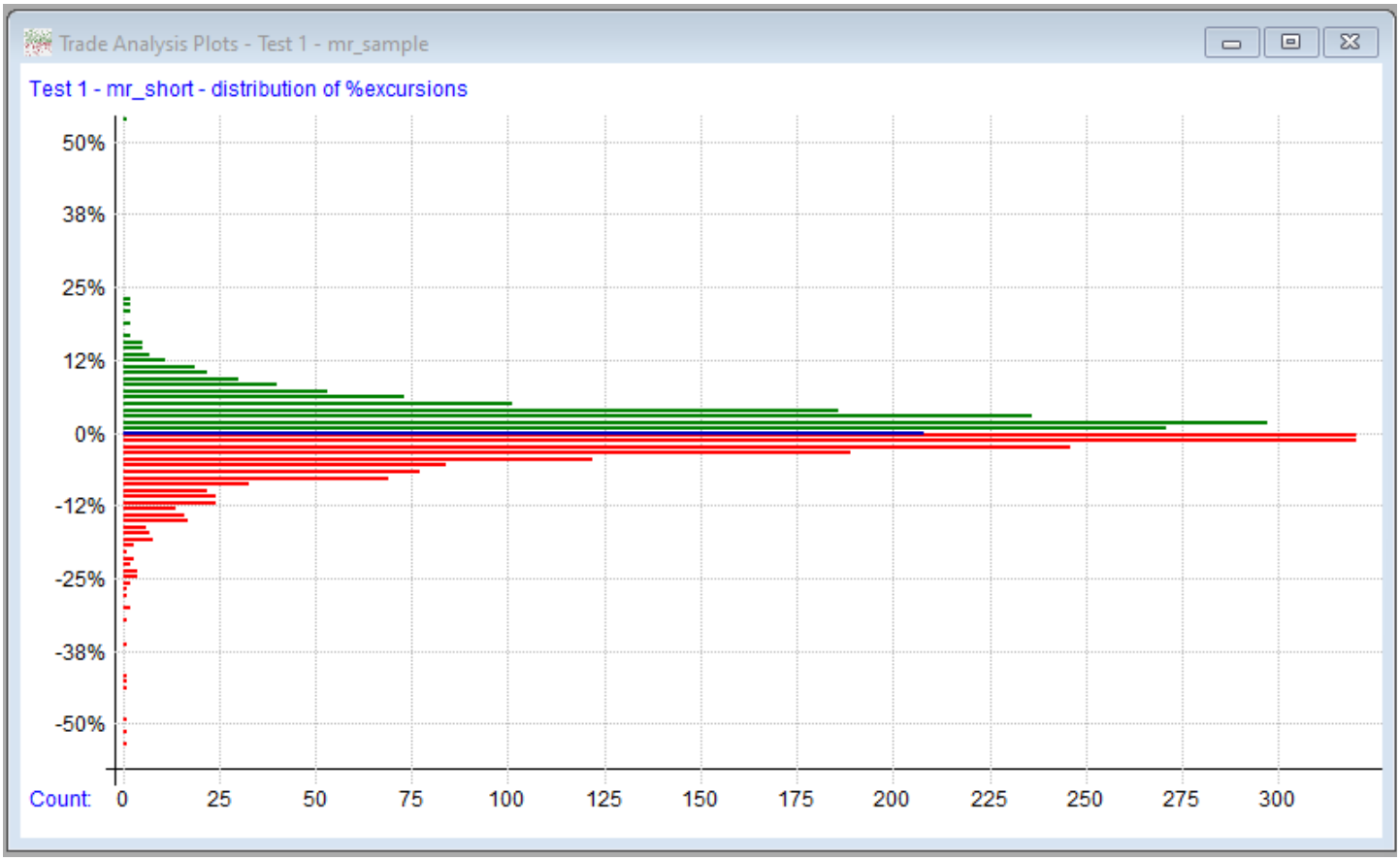


The upward slope of the linear regression line clearly shows that the more "overbought" the stock was prior to entry, the higher the expectancy of the trade. This is a real edge.

Profit Distribution plots a horizontal line for each range of % gains, something like a sideways "bell curve". If a strategy has "fat tails" they'll be visible on this plot (this one has the wide, thin tails typical of short-term mean reversion systems).



MAE/MFE Distribution is similar to the above, but rather than plotting closed-trade gains, it plots maximum open-position drawdowns and run-ups, also known as "adverse and favorable excursions".



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

Cumulative Profit is similar to a typical equity curve, but is derived from individual trades rather than daily net change of equity. Green dots show new highs of the profit line.

Equal Count Bins is a plot type designed for quick visual analysis of potential setup-day edge factors.

Here is an example using the same **C / EMA(C, 5)** formula as in the sorted scatter plot above:

Use the *Equal Count Bins* field in the Plot Options dialog to specify how many bins to use.

If *Log Analysis Stats* is checked when this plot is opened or refreshed, it will provide output like this:

Log - (untitled)*

```
-------------------------------------------------------------------------------
Test 1 - mr_short - average %gain, sorted by C / EMA(C, 5) - 160 trades per bin
-------------------------------------------------------------------------------
Bin            Count          X Min          X Max          Avg Trade
---            -----          -----          -----          ---------
1              160            1.01665        1.03196        -0.0114557
2              160            1.03198        1.03683        -0.00108835
3              160            1.03684        1.04136        -0.0067592
4              160            1.04141        1.04606        -0.0016608
5              160            1.04607        1.05108        0.00316169
6              160            1.05119        1.05675        0.003392
7              160            1.05699        1.06459        -0.00367212
8              160            1.06462        1.0791         0.00465721
9              160            1.07913        1.09917        0.0112652
10             160            1.09934        1.26064        0.0169126
```

Equal Range Bins uses the same concept as the above, but the bin contents are determined by value range rather than count.

Here's the same formula plotted with a bin for every 0.02 increment of the formula value:

Here are the stats for that plot:

Log - (untitled)*

```
-------------------------------------------------------------------------------
Test 1 - mr_short - average %gain, sorted by C / EMA(C, 5) - bin interval = 0.02
-------------------------------------------------------------------------------
Bin            Count          X Min          X Max          Avg Trade
---            -----          -----          -----          ---------
1              2              1              1.02           0.0356494
2              423            1.02           1.04           -0.0069499
3              616            1.04           1.06           -0.000542306
4              252            1.06           1.08           0.00409334
5              148            1.08           1.1            0.0125044
6              72             1.1            1.12           0.0195323
7              40             1.12           1.14           0.00718633
8              16             1.14           1.16           0.0297369
9              18             1.16           1.18           0.0173741
10             7              1.18           1.2            0.0174578
11             4              1.2            1.22           -0.01536
12             1              1.22           1.24           0.0496743
13             2              1.26           1.28           0.0255453
```

Factor Scan automates the kind of analysis shown in the last two plot types: instead of trying one formula at a time in the Formula field, you tag any number of Trades-section formulas with *{factor}* and a single scan finds the most profitable cutoff for every one of them, ranks the factors in a side panel, and plots the selected factor's cutoff profile or its kept and discarded trades over time. See **Factor Scan** for details.

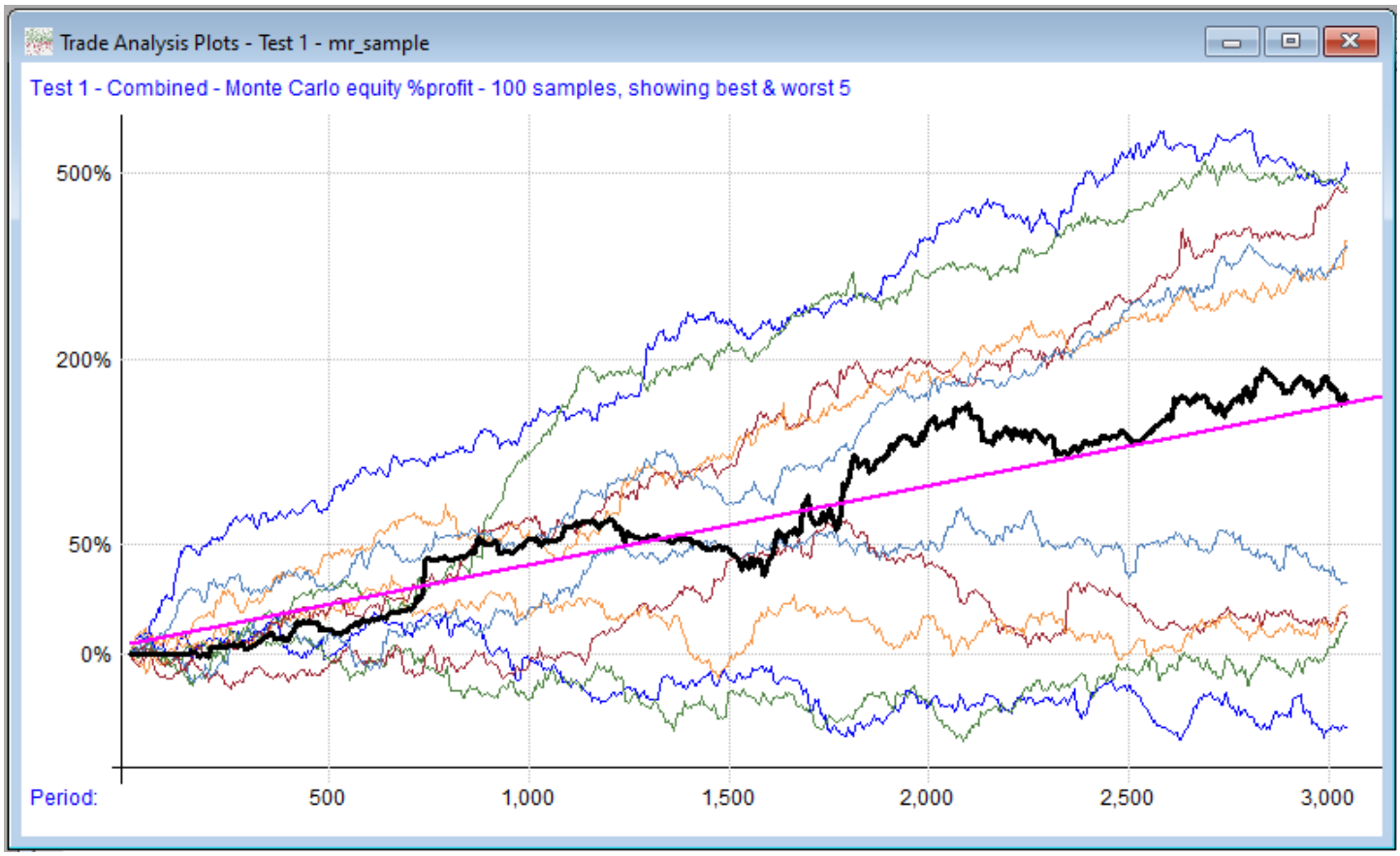
The last four plot types are the **Monte Carlo Analysis** plots.

The idea here is to randomly re-order either the individual trades or the daily percent changes in account value many times to see what the possible range of outcome might have been. RealTest uses "random selection with replacement".

Imagine putting each trade or account change value in a bucket. Pick one, record it as the first value, put it back, shuffle the bucket. Then pick the second value, put it back, shuffle the bucket, and so on. Continue until you've picked the same number of values (trades or stat periods) as the test had.

Repeat that entire process hundreds or even thousands of times. Plot the best and worst curves that emerge, along with the original backtest curve.

Here's an example from the mr_sample backtest showing randomized daily equity changes:



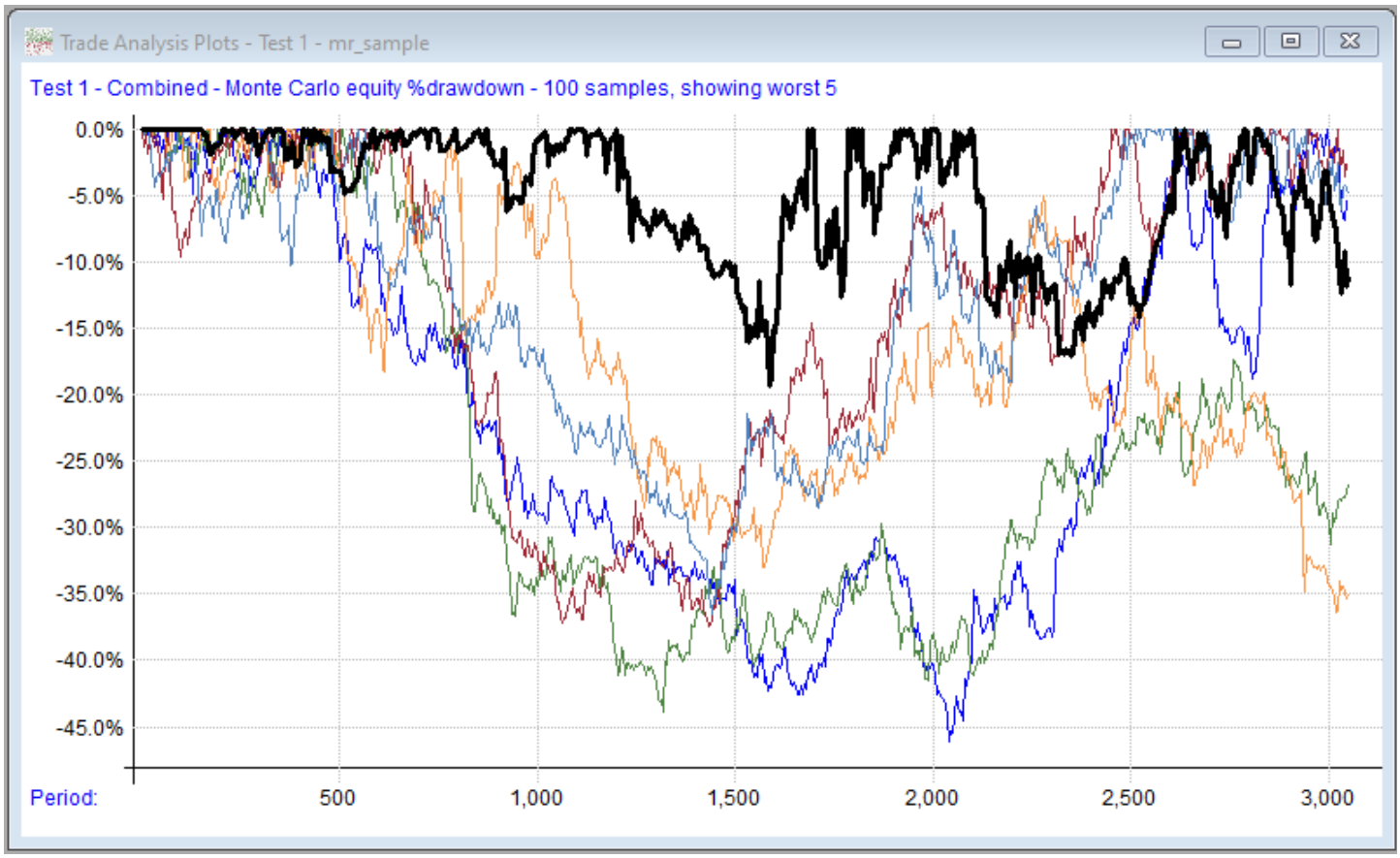
Keep in mind that every time you refresh this plot, it will look different, because a new set of random values are selected.

Here is the log output for Monte Carlo analysis:

Log - (untitled)*

```
-----------------------------------------------------------------
Monte Carlo Analysis
-----------------------------------------------------------------
Percentile     Net Profit     CAR            Max Drawdown   CAR/MaxDD
1%             12.36%         0.97%          -37.90%        0.03
5%             42.49%         2.97%          -31.65%        0.09
10%            60.88%         4.01%          -28.22%        0.14
20%            96.60%         5.75%          -23.81%        0.24
50%            175.37%        8.74%          -18.92%        0.46
80%            269.86%        11.42%         -14.50%        0.79
90%            331.47%        12.85%         -11.29%        1.14
95%            365.35%        13.56%         -9.29%         1.46
99%            531.25%        16.46%         -8.74%         1.88

backtest       156.69%        8.53%          -19.71%        0.43

Probability of loss: 1.0%
Probability of worse drawdown: 45.0%
Backtest result percentile: 42%
```

Here is an example of a Monte Carlo equity drawdowns plot:



Again, this is not the same set of randomized equity curves as the first plot above -- every time a plot is drawn or refreshed the randomization is recalculated.

This plot looks alarming but remember that these are the worst 5 drawdowns from 100 randomizations of the daily results, so while it's certainly possible to have a 45% drawdown in this mean-reversion strategy, it's not highly probable.

6.7.4.1. Factor Scan

Factor Scan is a **Trade Plot** display type that batch-tests a whole list of setup-day indicators - "factors" - against a test's trade list, and ranks them by how well each one would have filtered the trades. It automates what the Equal Count Bins and Equal Range Bins plots let you do one formula at a time: instead of typing an expression into the Formula field and eyeballing the bars, you tag any number of Trades-section formulas with *{factor}*, and one scan finds the most profitable cutoff for every one of them.

The intended workflow is to run a loosely-defined "backbone" version of a strategy - minimal filters, simple fixed sizing, as many trades as practical - then use Factor Scan to discover which conditions, applied at entry time, would most improve the average trade. Promising rows translate directly into **EntrySetup** conditions.

The idea for this feature came from **Dave Mabe**, and it is included in RealTest with his kind permission.

Tagging Factors

A factor is any numeric formula in a **Trades section** (in your strategy script, an included file, or the default Trades.rts) marked with the *{factor}* attribute:

```
Trades:
    Bars:           {#}             T.Bars
    PctGain:        {%-2}           T.NetPct

    f_rsi14:        {factor}        RSI(14)
    f_atr_pct:      {factor}        ATR(14) / C
    f_dist50:       {factor}        C / Avg(C, 50) - 1
    f_vol_ratio:    {factor}        V / Avg(V, 20)
    f_range252:     {factor}        (C - Lowest(L, 252)) / (Highest(H, 252) -
```

```
Lowest(L, 252))
```

- *{factor}* can stand alone or combine with **format attributes**: *{factor, format: percent, places: 2}*.
- A *{factor}* formula is evaluated **as of the trade's setup** - the last completed bar before the entry - everywhere it appears: in the scan, and as a Trades window column. No *[1]* or *[T.Bars+1]* offset is needed to avoid look-ahead.
- Untagged Trades-section formulas keep their usual behavior: they see the trade's exit bar as the current bar. To show both readings of the same indicator, define it twice - once tagged, once not.
- Because tagged columns show exactly the values the scan used, the Trades window and its CSV export are the scan's ground truth: a trade list saved with factor columns is a complete research dataset (outcomes plus setup-day factors) for outside analysis.
- String-valued formulas are ignored by the scan.

Prefer scale-free factors - ratios, oscillators, percent distances, positions in range - over raw levels. A price-denominated factor (a moving average, a band, *C* itself) mostly measures share price, so a battery full of them will "discover" the same price effect many times over. The Ovlp column (below) exposes this, but a well-chosen battery avoids it.

Running a Scan

Open Trade Plots for a test as usual (right-click it in a **Results Window** and choose Show Trade Plots), then select the **Factor Scan** tab, press **G**, or choose it from the Plot menu or the Plot Type option.

The factors come from the **currently active** Trades section - the same source as the Trades window columns - not from the script that originally ran the test. This means you can apply any script containing tagged factors and immediately scan any test in your results list, including old ones: run the backbone test once, then iterate on the factor list freely. Applying a changed Trades section automatically rescans any open Factor Scan plots.

The first scan evaluates every factor for every trade (a progress message shows each factor; Esc stops early and keeps what finished). The values are then buffered for the window, so everything after that first pass - changing the trade value mode, clicking rows, adjusting options - is instant. Factors are only re-evaluated when the Trades section is re-applied.

If a trade's symbol isn't in the currently loaded data, or the factor has no value on the setup bar (for example, not enough history), that trade is skipped for that factor and counted in the title bar and log.

If the active Trades section has no *{factor}* items, or the test has too few trades to scan, the plot area simply says so - the window still opens and the other plot types work as usual.

Reading the Results

The left panel lists every factor, best first:

Column	Meaning
Rank	The factor's place in the ranking by Edge (1 = best). It stays with the factor when the panel is sorted by another column.
Factor	The formula's name from the Trades section.
Dir	Which side of the cutoff is kept: >= keeps trades with values at or above the cutoff, <= at or below.
Cutoff	The factor value at the best split point found.
Pctl	The same cutoff as a percentile of the factor's ranked values - *20%* with <= means "keep the bottom fifth of trades by this factor." This is the portable way to state a threshold; the raw value is just where that percentile landed in this sample.
Kept	Trades retained / trades that had a value for this factor. A trailing * means the cutoff is pinned at the retention floor (see Min Kept below): the average was still improving when the floor stopped it, so read the row as "more extreme is better, as far as could be measured," not as a precise optimum.

Total	Total gain of the kept trades (Kept × Avg), in the current trade value mode. With fixed-size trades this is what the filtered trade list would have made. Sort by it for a different perspective: a cutoff with a high Avg may keep so few trades that it makes less in total than a milder one.
Avg	Average gain per kept trade, in the current trade value mode.
Edge	Avg minus the baseline (the average over all of this factor's evaluated trades). Rows are ranked by Edge - how much the filter improves on doing nothing.
OOS	An out-of-sample check: the cutoff is re-fit using only the first (chronological) half of the trades, then scored on the second half alone. A large Edge with an OOS near or below the baseline suggests the edge did not persist. *n/a* appears when either half has too few trades.
Ovlp	How much this factor's kept trades overlap those of the factor ranked just above it (the next lower Rank). High overlap means the two factors selected largely the same trades - one finding echoed, not two independent ones. The top-ranked factor has no Ovlp value.

Click a column heading to sort the panel by that column, and click it again to reverse the order; shift-click another heading to add it as a secondary sort, as in the other list windows. Sorting by Factor is useful for comparing scans: run the same factors over different periods or strategy variations and see where each one ranks in each scan. Sorting by OOS brings the factors whose edge held up out of sample to one end. The sort stays in effect when the scan reruns and carries over to newly opened plot windows; each RealTest session starts in rank order.

Right-click the panel for **Copy List to Clipboard**, **Copy Row to Clipboard** and **Save List as CSV File...**, as in the other list windows. They copy or save the table as shown, in its current order; for the Plateau and Skipped columns as well, log the table (below).

Click a row - or press **Up/Down** - to show that factor in the plot pane: its cutoff profile, or its kept and discarded trades over time (see below). After a sort, the new top row is selected. (In Factor Scan mode the arrow keys move the factor selection, in the order shown; in every other plot mode they move to the next or previous test as usual.)

The Cutoff Profile Plot

The plot shows the selected row's whole story rather than just its single best number:

- The **green line** traces the average gain of the kept trades at every possible cutoff. For a <= row, the value at x = 30 is the average gain of the bottom 30% of trades by that factor. At the keep-everything end the line meets the baseline average (also drawn as a dotted horizontal line).
- The **magenta bar** marks the chosen cutoff from the table.
- A smooth, gradual curve indicates a broad, well-behaved effect; a narrow spike at one threshold is more likely noise.

The **Sort By** (X axis) choice controls the horizontal scale:

- **Percentile** - cutoffs from 0 to 100 percent of trades. Press **I**.
- **Formula** - the raw factor values. Press **F**. Note that a factor with outliers will compress most of the curve into one end of a value axis; that distortion is itself informative, and is why the percentile scale is usually the better read.

The third choice, **Trade Number** (press **T**), switches to the time view described next. The crosshair (X) works in every mode.

Kept vs Discarded Over Time

With Sort By set to **Trade Number** (the default), the plot shows the selected row's split through time instead of across cutoffs. The trades are taken in the order they were entered, and three running totals of the scoring metric are drawn:

- The **green line** adds up the trades the row's cutoff keeps.
- The **red line** adds up the trades it discards.
- The **gray line** adds up all of them (green plus red).
- A dotted **blue line** marks the split between the first and second half of the trade list - the same

split the OOS column uses.

The X axis is labeled with entry dates. The gains are summed rather than compounded, like a fixed-size equity curve, which matches the scan's average-trade objective.

This view answers a question the table can't: **when** did the edge appear? Because the cutoff was chosen using all the trades, the green and red lines always finish apart - the ending gap proves nothing. What matters is the shape. A factor whose lines diverge early and keep diverging steadily is more trustworthy than one where they track each other for most of the test and separate only in one period. When choosing between rows with similar Edge, stepping through them with Up/Down and comparing these curves is often the quickest way to decide; the OOS column summarizes the same question with a single number.

Keep in mind that the two groups usually differ in size: at the default Min Kept of 20%, the discarded group can have four times as many trades as the kept group. If the discarded trades are still profitable on average, the red line can rise faster than the green one even though each discarded trade is worse. Compare the slopes with that in mind, or raise Min Kept % to make the groups closer in size.

The Scoring Metric

The scan maximizes the mean of whatever the **Trade Value** mode selects, so the metric is chosen the same way as in the other plot types:

- **Dollars** (*$*) - net profit per trade after commission and slippage.
- **Trade Percent** (%) - net profit as a percent of position size.
- **Alloc Percent** (^) - net profit as a percent of account equity at entry.

The plot title names the metric in use ("average trade %gain", "average alloc %gain", "average $gain"; "cumulative ..." in the time view). Changing it rescans instantly from the buffered values. Trade Percent is the usual choice for entry-filter research; note that Dollars can be dominated by a few large positions, and Alloc Percent weights trades by their allocation size.

Factor Scan Min Kept %

This plot option (default 20) is the retention floor: the smallest percentage of trades a cutoff is allowed to keep. It prevents the scan from "winning" by keeping a handful of lucky outliers, and it bounds how far the profile curve extends.

Rows marked * are clamped at this floor. Lowering the floor (say to 10 or 5) lets the scan chase more extreme cutoffs - often with dramatic effect on the rankings - and watching whether OOS holds up as you descend is a useful robustness test. The same floor governs the out-of-sample fitting, so the whole report stays consistent at whatever level you set.

Logging the Table

Press **A** (Log Analysis Stats) and refresh to write the full table to the Log window, rows in the panel's current order. The log adds two columns not shown in the panel:

- **Plateau** - the contiguous range of cutoff percentiles that retain at least 90% of the row's edge. A wide plateau means the result is not sensitive to the exact threshold; a knife-edge plateau is a warning sign.
- **Skipped** - trades that had no value for that factor.

The log table is tab-delimited and pastes cleanly into a spreadsheet.

Multi-Strategy Tests

Factor Scan works on multi-strategy tests: the strategy panel appears to the left of the factor panel (each panel edge has its own draggable splitter), and selecting a strategy - or Combined - rescans instantly, since the buffered factor values cover every trade. Keep in mind that in a combined test each trade's size depends on what all strategies were doing at the time, so per-strategy scans are the cleaner read; the Combined view is best treated as a survey.

Using What You Find

A promising row converts directly into a strategy condition:

```
EntrySetup:  {existing setup conditions} and C / Avg(C, 50) - 1 <= -0.0675
```

Some suggestions for keeping the research honest:

- **Use the raw cutoff value in the condition, but judge the row by its percentile.** "The bottom 20% by distance from the 50-day average" is the finding; *-0.0675* is just where that percentile fell in this sample. The Pctl and Plateau columns tell you whether the threshold sits on a broad effect or a knife-edge; the raw value is simply how the rule gets written.
- **For thresholds that stay percentile-based, make the factor itself a rank.** A factor defined as *PercentRank(C / Avg(C, 50) - 1, 252)* ranks each bar against the symbol's own recent history, so the scan's cutoffs are already percentiles and port into *EntrySetup* unchanged. *#PercentRank* does the same across the universe on each bar. (Each ranks against a different population than the scan's pooled trade sample - all are legitimate, and rank-style factors are inherently scale-free, which makes them good battery material in general.)
- **Check OOS and the plateau before trusting Edge.** A ranked list of many factors is a multiple-comparison exercise - something will always look good in-sample.
- **Watch Ovlp for echoes.** Five highly-overlapped rows are one idea, not five confirmations.
- **Re-test properly.** After adding a condition to the strategy, run the real backtest - with actual sizing, limits, and compounding - rather than assuming the scan's average carries over.

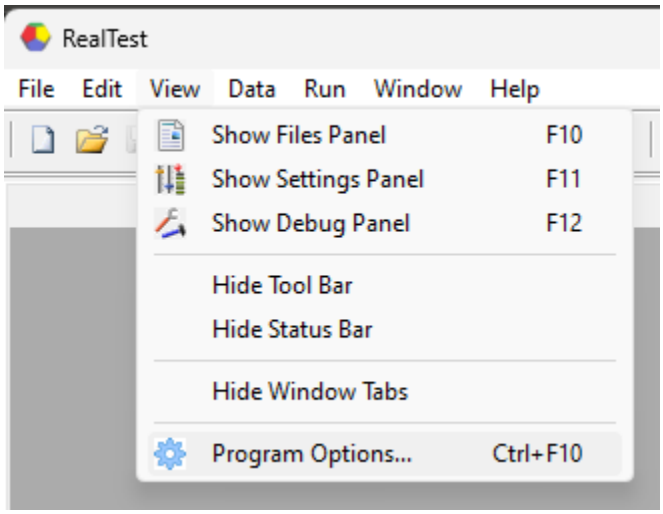
Keyboard Summary (Factor Scan mode)

Key	Action
G	Switch to Factor Scan
Up / Down	Select the previous / next factor in the panel
F / I / T	X axis: cutoff profile by factor value / cutoff profile by percentile / kept vs discarded over time
$ / % / ^	Scoring metric: dollars / trade percent / alloc percent
A	Log the ranked table on the next refresh
P	Show or hide the strategy panel (multi-strategy tests)
X	Crosshair
O	Options

See also **Trade Plots and Analysis**, **Trades Section** and **Output Format Specification**.

6.8. Program Options Dialog

The Program Options dialog is accessible via the **View Menu**, or by pressing Ctrl+F10:

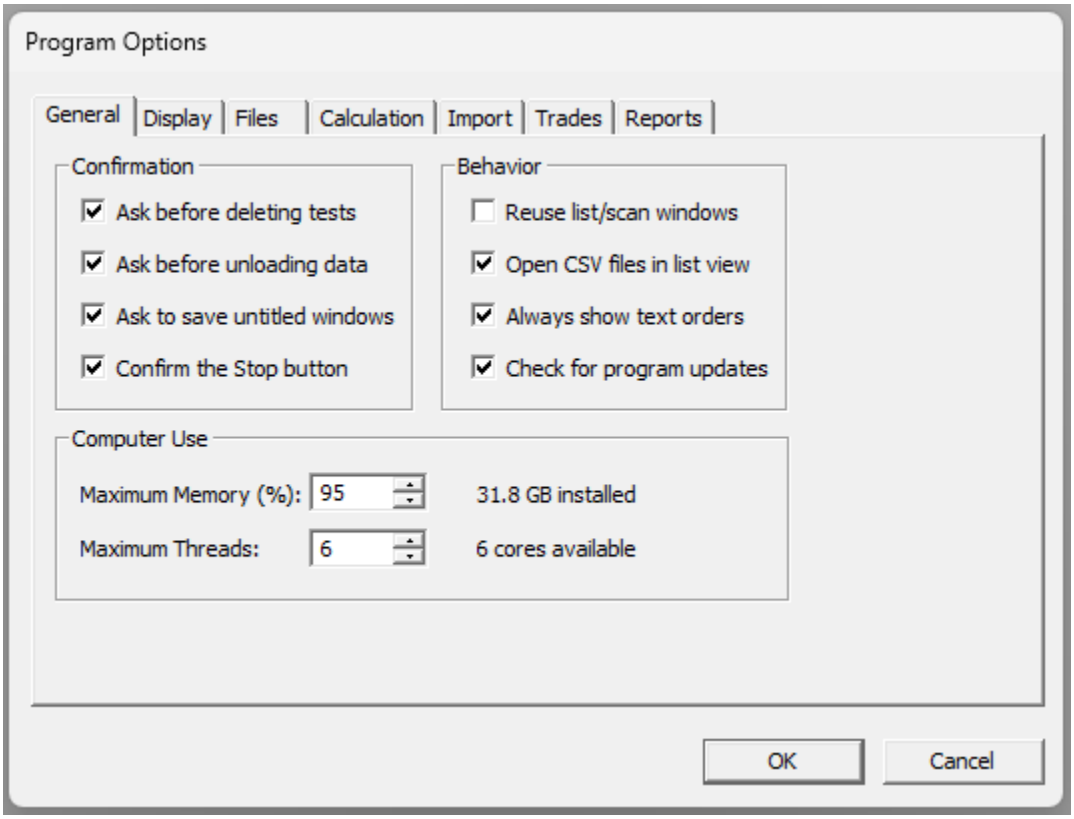


The options are organized into seven tabs: General, Display, Files, Calculation, Import, Trades, and Reports. The tab you were last using is remembered and reselected the next time the dialog is opened. Changes take effect when you click OK.

General

Confirmation prompts, window behavior, and computer resource limits.

Each of the confirmation prompts controlled by the "Ask" and "Confirm" options below also includes a "Don't ask me again" checkbox. Checking it when the prompt appears clears the corresponding option here, so you never have to visit this dialog just to silence a prompt.



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

- **Confirm the Stop button**

  Specifies whether RealTest should pop up a dialog box for confirmation before aborting a script when the "Stop" button is pressed.

- **Reuse list/scan windows**

  Allows scans and other tabular data output to optionally reuse any currently open window of the relevant type rather than always creating a new window.

- **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 is 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 program 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.

- **Maximum Memory**

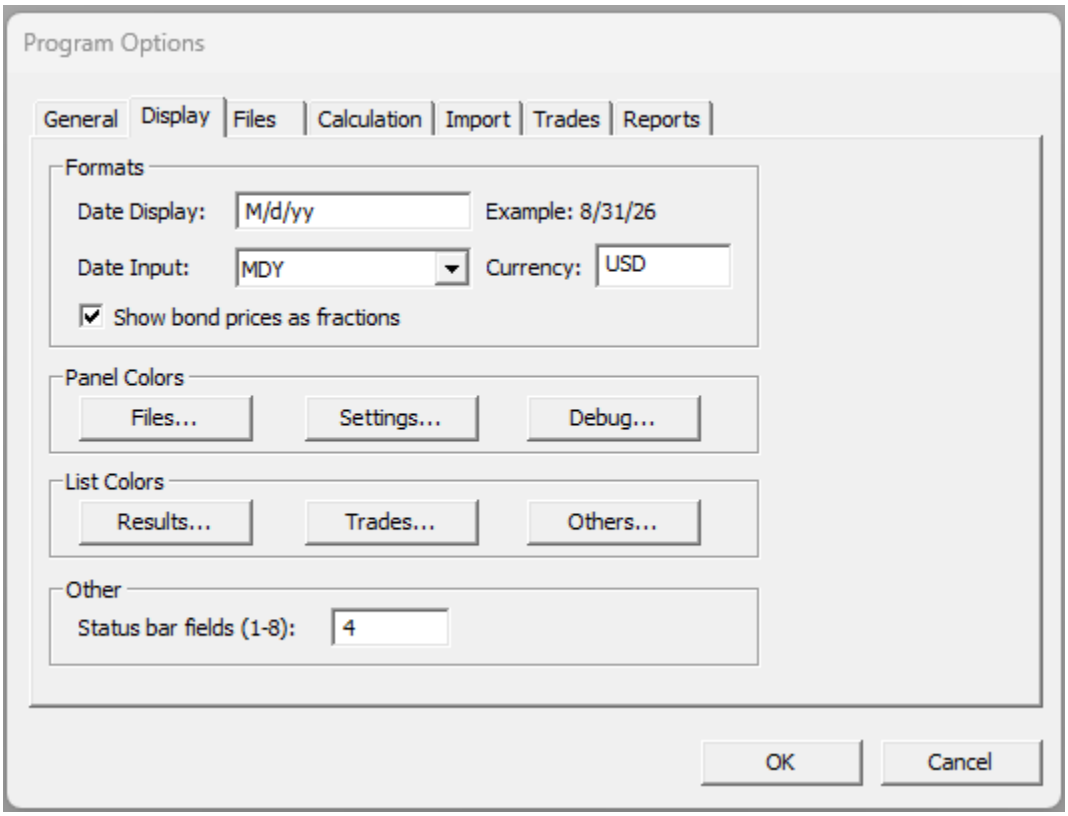
  Specifies a threshold (percent of installed RAM) above which RealTest will stop trying to allocate memory and instead will give a warning. Program stability is not guaranteed if you try to use more than 100% of installed RAM, i.e., make excessive use of virtual memory page swapping. This setting also governs the **out-of-memory check** that runs before Data section calculation: setting it above 100 allows runs whose projected memory use exceeds what can currently be committed, provided the page file can grow.

- **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.

Display

Date and currency formats, panel and list background colors, and other display preferences.



- **Date Display**

  Specifies how dates are displayed in the user interface, scans, trade lists, etc. See **Date Format Pictures** for details about the format specification string. Use the adjacent example output to test your format as you type it. Can be overridden by the **DateDisplay** setting in a script.

- **Date Input**

  RealTest can parse dates in any common format. For formats where *day* and *month* precede *year*, this setting governs whether *day* precedes *month* or vice versa. Can be overridden by the

DateInput setting in a script.

- **Currency**

  Specifies the default base currency of the trading account. Can be overridden per script by the **Currency** setting, or per run by the Currency field on the **Settings Panel**.

- **Show bond prices as fractions**

  When checked, the prices of bond and note instruments are displayed in fractional (32nds) notation rather than as plain decimals.

- **Panel Colors**

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

- **List Colors**

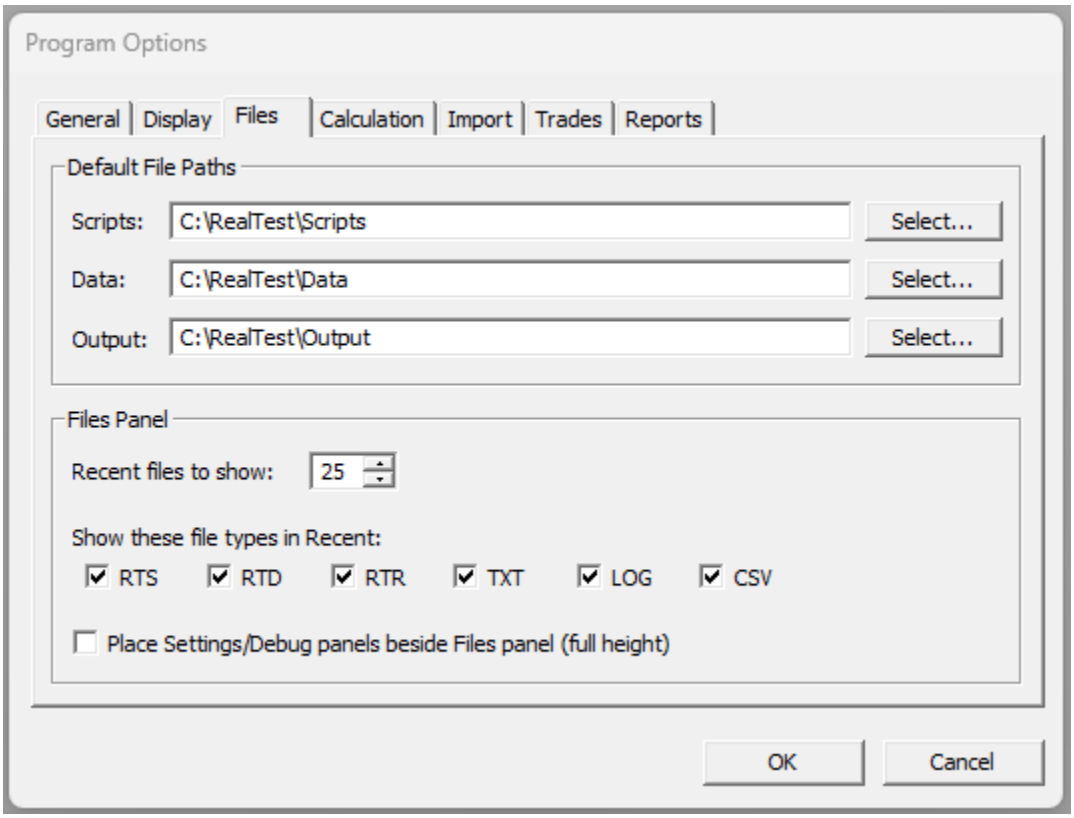
  Lets you change the background color of **Results Windows**, **Trades Windows** and other **List Windows**.

- **Status bar fields**

  Specifies how many of the available information fields (1 to 8) to show in the status bar at the bottom of the main window.

Files

Default file paths and **Files Panel** settings.



- **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 Syntax** 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.

- **Recent files to show**

  Sets how many entries (1 to 100) appear in the **Files Panel**'s Recent section and on the **File Menu**'s recent list (the menu shows at most nine).

- **Show these file types in Recent**

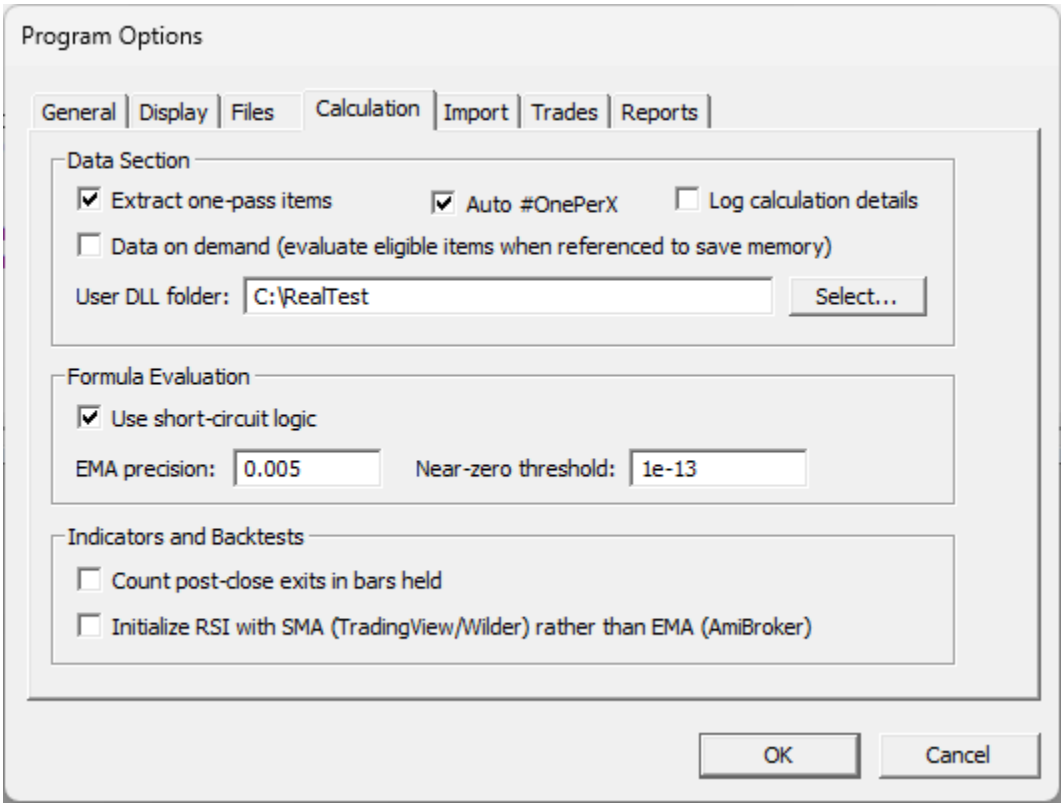
  Filters the Recent section by file type (RTS scripts, RTD data, RTR results, TXT, LOG, and CSV). Unchecking a type hides those files without removing them; re-checking brings them back. Files of other types always appear.

- **Place Settings/Debug panels beside Files panel (full height)**

  When checked, the Files panel occupies the full height of the window's left edge and the **Settings** and **Debug** panels are placed in a column to its right, above the workspace, rather than stacked below the Files panel.

Calculation

Settings that affect how **Data Section** items, formulas, and indicators are calculated.



- **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.

- **Data on demand**

  When checked, eligible Data Section items are evaluated each time they are referenced rather

than being calculated up front and stored, letting scripts whose stored data would not fit in memory run in a fraction of the space (more slowly). See **Data On Demand** for details, including which items remain stored.

- **User DLL folder**

  Specifies the folder RealTest searches for user-supplied DLLs used by custom data calculations.

- **Use short-circuit logic**

  When selected (default), all formula evaluation throughout RealTest employs **short-circuit optimization**. When unchecked, every term of every formula is evaluated even when not logically necessary.

- **EMA precision**

  The convergence threshold used to decide how many warm-up bars are needed before an exponential moving average is considered fully seeded. The default (0.005) suits virtually all scripts.

- **Near-zero threshold**

  Absolute values smaller than this are treated as exactly zero during formula evaluation, preventing spurious results from floating-point rounding. The default (1e-13) rarely needs changing.

- **Count post-close exits in bars held**

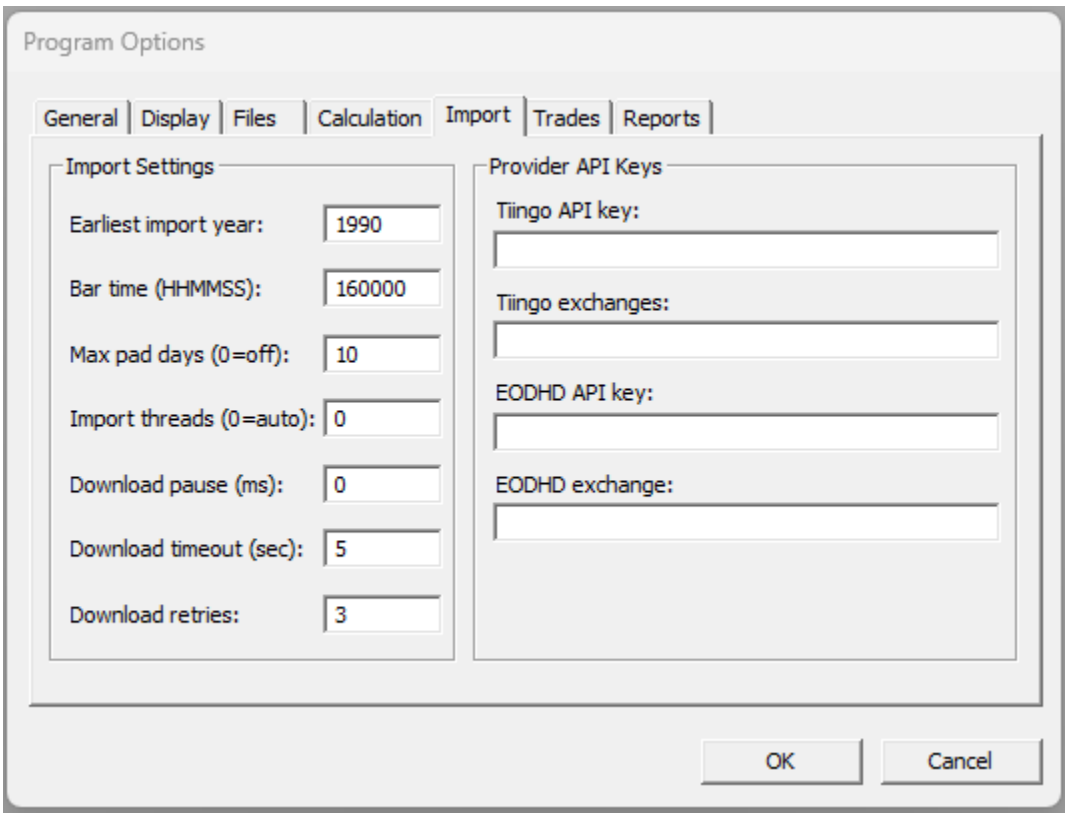
  When checked, a position that exits in post-close processing on its entry bar counts as one bar held rather than zero.

- **Initialize RSI with SMA (TradingView/Wilder) rather than EMA (AmiBroker)**

  Selects how **RSI** and **RsiF** seed Wilder's smoothing at the start of a symbol's history. The two methods differ only during the warm-up region (roughly the first 3-4x the period in bars) and converge thereafter. See **RSI** for details.

Import

Settings that control bar-data **imports** and downloads, including data-provider API keys. These were previously available only by editing RealTest.ini.



- **Earliest import year**

  Bars dated before this year are skipped on import, keeping very old history out of your data file.

Default 1990.

- **Bar time (HHMMSS)**

  The time-of-day stamp assigned to imported daily bars, in HHMMSS form. Default 160000 (4:00:00 PM).

- **Max pad days (0=off)**

  Limits trailing data padding to symbols whose last real bar is within this many days of the data's latest date (0 disables the trailing pad). See **PadTrailMaxDays**.

- **Import threads (0=auto)**

  How many parallel download threads to use when importing from an online provider. 0 lets RealTest choose.

- **Download pause (ms)**

  An optional delay between successive download requests, in milliseconds, to stay within a provider's rate limits. 0 means no pause.

- **Download timeout (sec)**

  How long to wait for each download request before giving up. Default 5.

- **Download retries**

  How many times to retry a failed download before reporting an error. Default 3.

- **Tiingo API key / Tiingo exchanges**

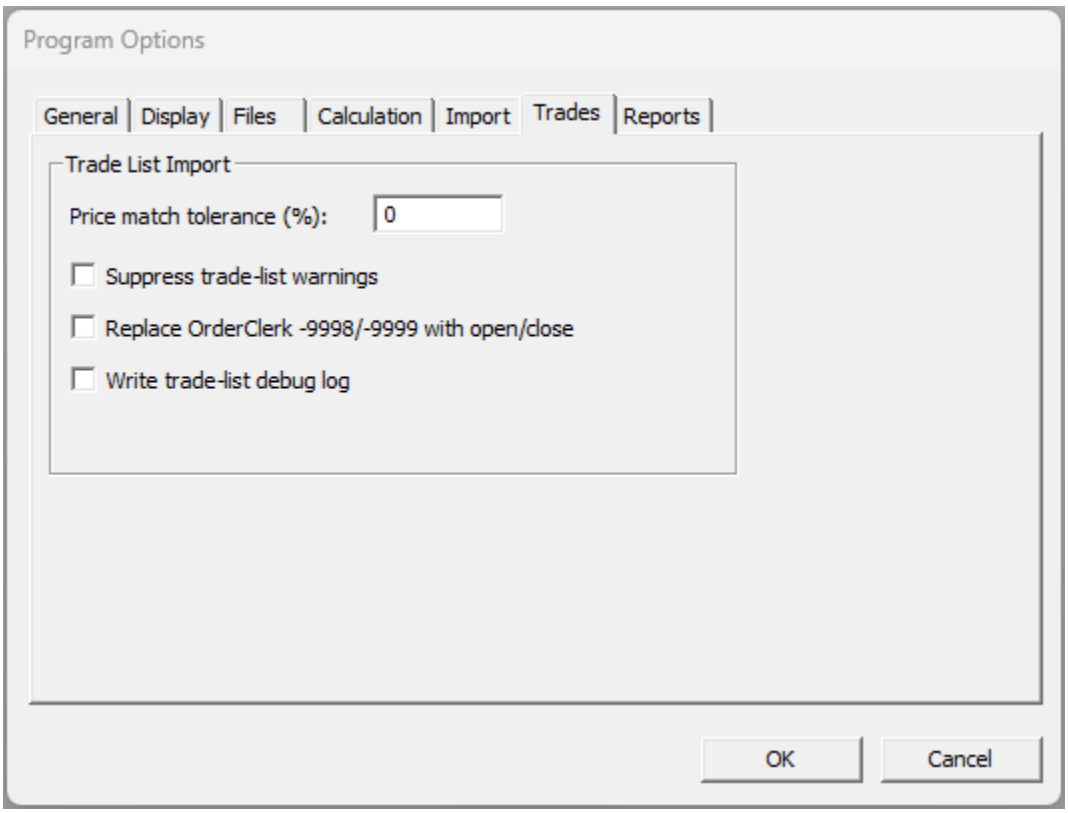
  Your **Tiingo** API key and, optionally, the list of exchanges to request when downloading.

- **EODHD API key / EODHD exchange**

  Your **EODHD** API key and, optionally, the exchange to request when downloading.

Trades

Options that govern how an **imported trade list** is validated against the bar data.



- **Price match tolerance (%)**

  The percentage tolerance allowed when matching an imported trade's price to the bar data during validation. 0 requires the price to fall within the bar's actual range.

- **Suppress trade-list warnings**

  When checked, pop-up warnings about trade-list import issues are suppressed. The messages still go to the log.

- **Replace OrderClerk -9998/-9999 with open/close**

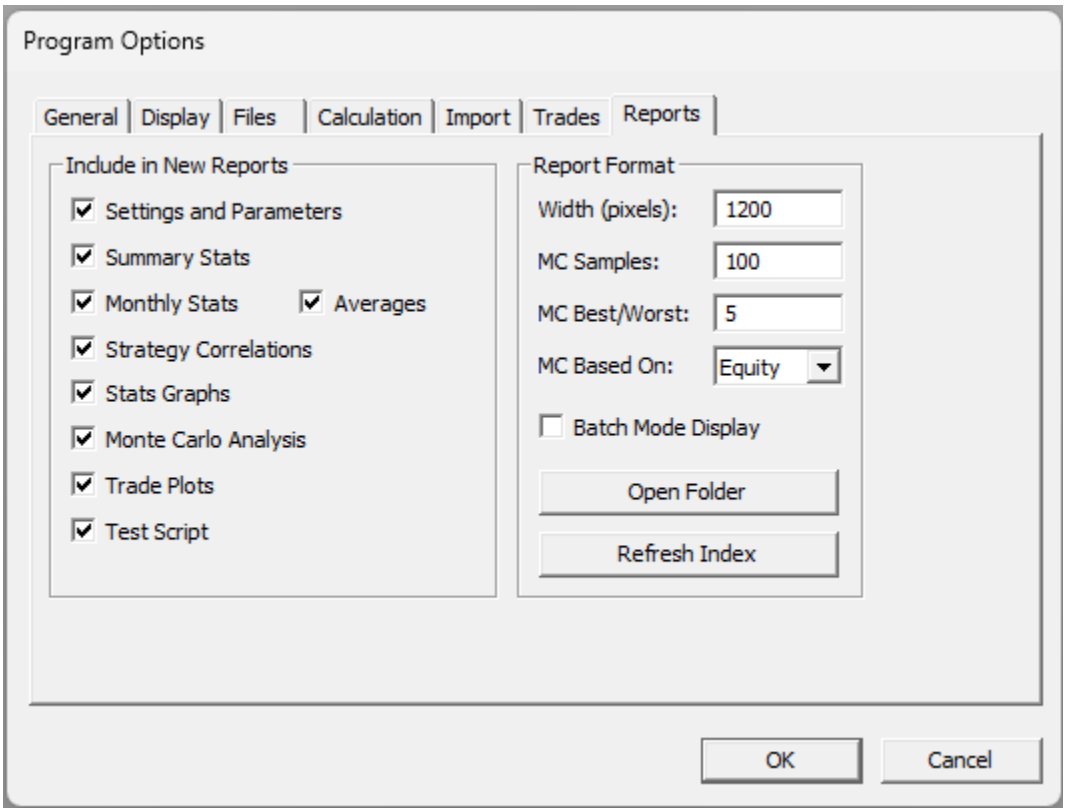
  OrderClerk writes the sentinel prices -9998 and -9999 to mean "filled at the open" and "filled at the close". When checked, these are replaced with the actual open or close price from the bar data as the trade list is imported.

- **Write trade-list debug log**

  Writes a detailed log file of the trade-list import process, useful for diagnosing matching problems.

Reports

Controls the content and format of newly created **Test Summary Reports**. Because reports are static web pages, these settings affect only reports created after they are changed.



- **Include in New Reports**

  Check the report sections to include or omit: Settings and Parameters, Summary Stats, Monthly Stats, Averages, Strategy Correlations, Stats Graphs, Monte Carlo Analysis, Trade Plots, and the Test Script.

- **Width**

  The maximum report width, in pixels, written into the generated HTML.

- **MC Samples, MC Best/Worst, MC Based On**

  Options for the Monte Carlo analysis plots, used only when Monte Carlo Analysis is included.

- **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.

6.9. 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.

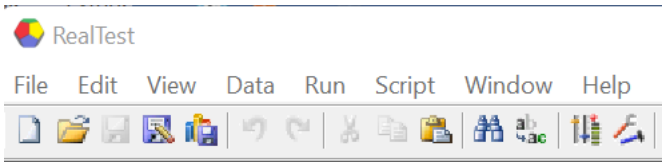
KEY	FUNCTION
Ctrl+F	Edit Find
Ctrl+Shift+F	Edit Find In Files
Ctrl+H	Edit Replace
Ctrl+N	File New
Ctrl+O	File Open
Ctrl+S	File Save
Ctrl+L	File Save All
Ctrl+P	Results Show Plot (also works in graph windows)
Ctrl+G	Results Show Graph (also double-click on results line or press Enter)
Ctrl+T	Results Show Trades (also works in graph windows)
Ctrl+R	Results Show Summary Report
Ctrl+/	Script Comment Toggle (also can click in left margin of line)
Ctrl+Break	Stops the execution of the currently running script
Ctrl+W	Open the Windows dialog (select or close multiple windows)
F1	Open the help file and show the page most relevant to what's under the cursor
F2	Show the script autocomplete list, or rename the selected test in a 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)
Ctrl+F9	Refresh calculations (ditto)
F10	Show or hide the Files Panel
F11	Show or hide the Settings Panel
F12	Show or hide the Debug 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")

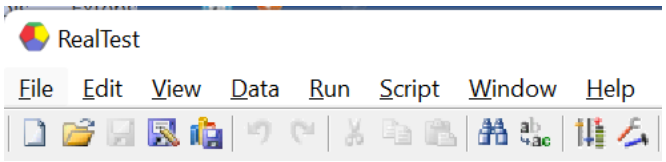
In addition to the above hotkeys, notice that most choices within the main menu and sub-menus have a letter assigned to them.

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

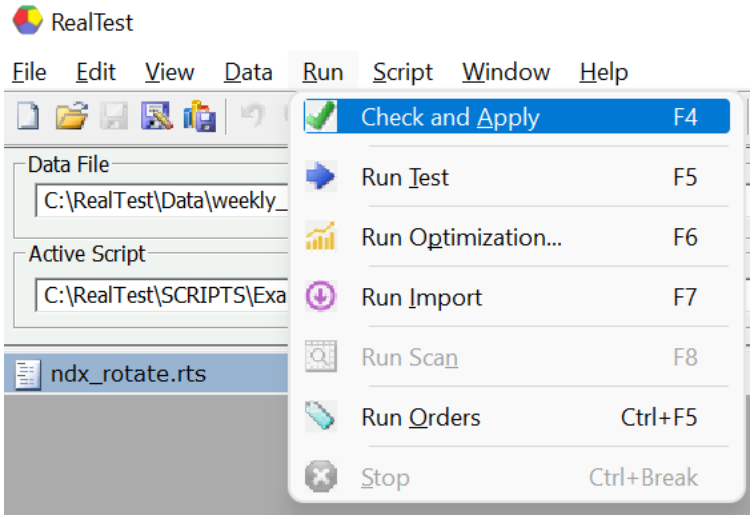
Main menu:



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

7. Importing Bar Data

Before running any scan or test, you import price and volume data into the RealTest binary format. These topics explain the import process and each supported data source.

- **How Importing Works** - the import process and how RealTest stores data
- **Norgate Data Import** - importing from Norgate Data Updater
- **CSI Futures Import** - importing futures from CSI
- **CSI Stocks Import** - importing stocks from CSI
- **CSV Import** - importing from generic CSV files
- **EODHD Import** - importing from EODHD
- **MetaStock Import** - importing from MetaStock data
- **Tiingo Import** - importing from Tiingo
- **Yahoo Import** - importing from Yahoo Finance
- **Multi-Source Import** - combining data from more than one source
- **Symbol Information Files** - adding per-symbol metadata
- **Futures Symbol Information** - per-contract futures metadata
- **Event List Files** - adding per-bar events such as earnings dates
- **Dividend and Constituency Events** - dividend and index-membership events

7.1. How Importing Works

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, monthly, quarterly and yearly bars from the daily bar data (importing larger bar sizes directly is not supported)
- 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**.

7.2. Norgate Data Import

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

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

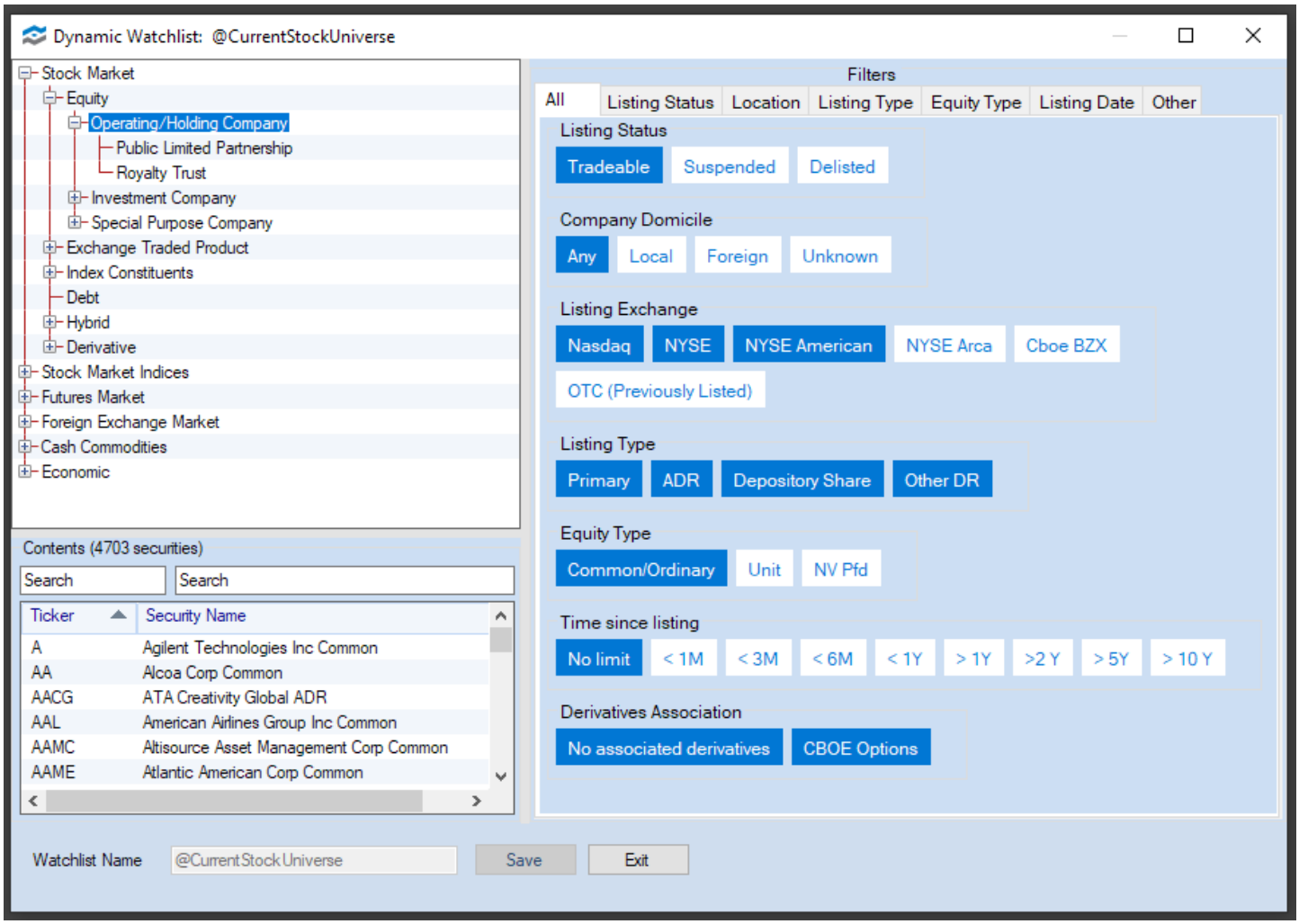
To import data from Norgate, the Norgate Data Updater (NDU) application must be currently running (open) on the same machine as RealTest.

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, the following Watchlist definition can be used for a "all US stocks" trading universe:



This only needs to be set up once, and is referenced in an 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
?Symbol	stock symbol	string
?Name	company name	string
?Type	security type	string
?Exchange	exchange name	string
?Domicile	security country	string
?Currency	security currency	string
?Sector	security sector	string
?Industry	security industry	string

?CII	corresponding industry index	string
InfoTRBC	Thomson Reuters Business Classification code	**described here**
InfoGICS	Global Industry Classification Standard code	**described here**
InfoShares	number of shares outstanding	number
InfoFloat	number of shares float	number
InfoDelist	delisting date or futures expiry	date
ListNum	import include list number for this symbol	number
PointValue	point value (futures - always 1.0 for stocks)	value
TickSize	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
```

Scan - 70 Items

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	AKLMQ_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 **cii_rotate.rts example script**.

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.

In addition to stocks, Norgate offers daily data for many of the global futures markets.

Norgate futures data includes two continuous-contract symbols per market, e.g. &ES (unadjusted) and &ES_CCB (back adjusted), plus all the historical individual contracts.

The **futures_calendar_spread.rts** and **futures_volume_rank.rts** **example scripts** show how to import and use Norgate futures data.

7.3. CSI Futures Import

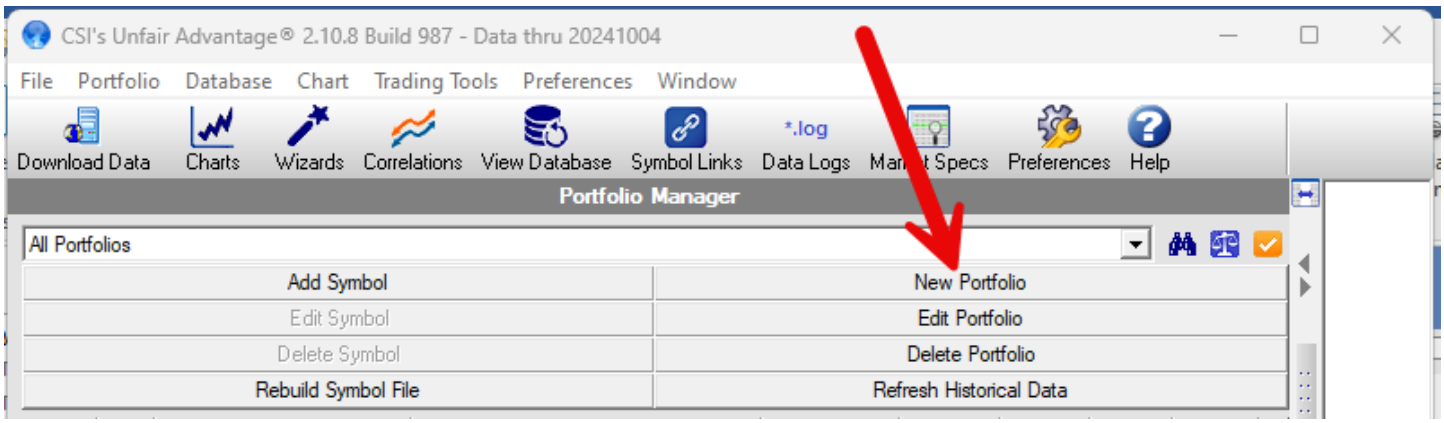
For end-of-day futures data, CSI has been the most respected source for many decades.

RealTest fully supports CSI's futures data and makes it easy to import and work with it.

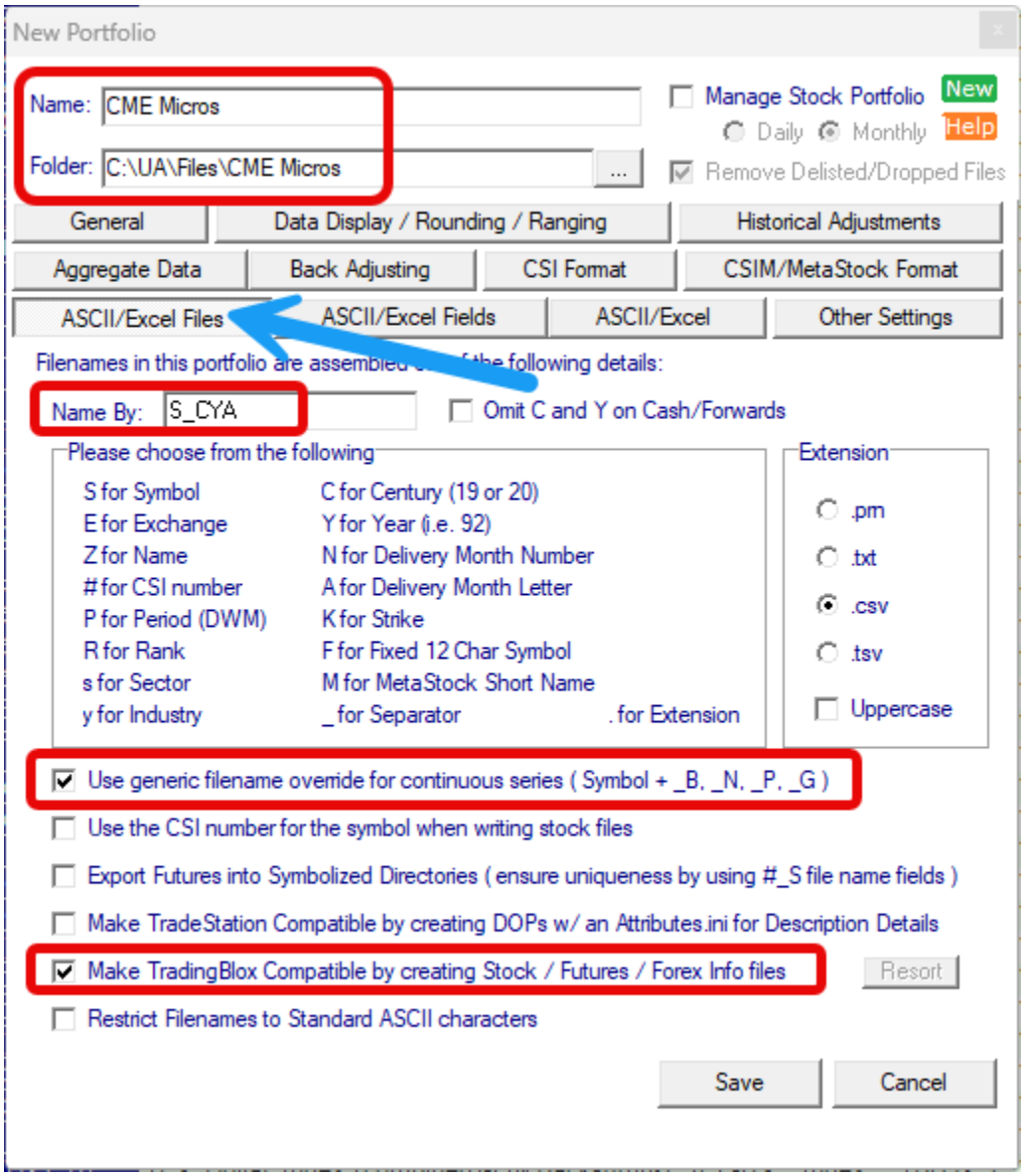
To use CSI data with RealTest, you must have downloaded and installed their **Unfair Advantage** software and subscribed to one of their futures data packages.

Within the *Unfair Advantage* program, create one or more new futures portfolios for use with RealTest.

1. Click on "New Portfolio":



2. Provide a name and folder location for the portfolio's CSV data files and set some properties of the CSV files to create:



It does not matter what the folder name and path are, just that you remember them so your RealTest script can point to it.

There are a large number of portfolio settings spread across 11 window tabs here.

The first two rows of tabs can be kept on their defaults.

The fields under "ASCII/Excel Files" (by which they mean CSV files) that are important for RealTest are:

a. the file names for individual contracts should use the S_CYA format, e.g. VX_2024Z

b. the "Make TradingBlox Compatible" box must be checked so that UA will create a "futuresinfo.txt" file

3. Specify the field-level details of the CSV export files:

The key here is that the field list includes at least the following items:

- D - date
- O - open
- H - high
- L - low
- C - close
- U - unadjusted close
- X - expiry date
- N - delivery number (delivery year and month)

It does not matter what order these appear in, just that they're all present.

If included, RealTest stores the unadjusted close price in the **Extra** field of each bar.

Because CSI's exported files include a header row, RealTest auto-detects these columns, so you normally do not need a **CSVFields** statement for a CSIFutures source. If an earlier CSV source in the same Import block declared explicit fields (which would otherwise carry forward), add *CSVFields: Auto* before the CSIFutures DataPath to restore auto-detection. If you must specify fields for a CSI source, name the unadjusted-close column *RealClose* (not Extra) so it reaches the **Extra** field.

If included, RealTest stores the expiry date and delivery number as**Event** records for the dates on which they changed (roll dates).

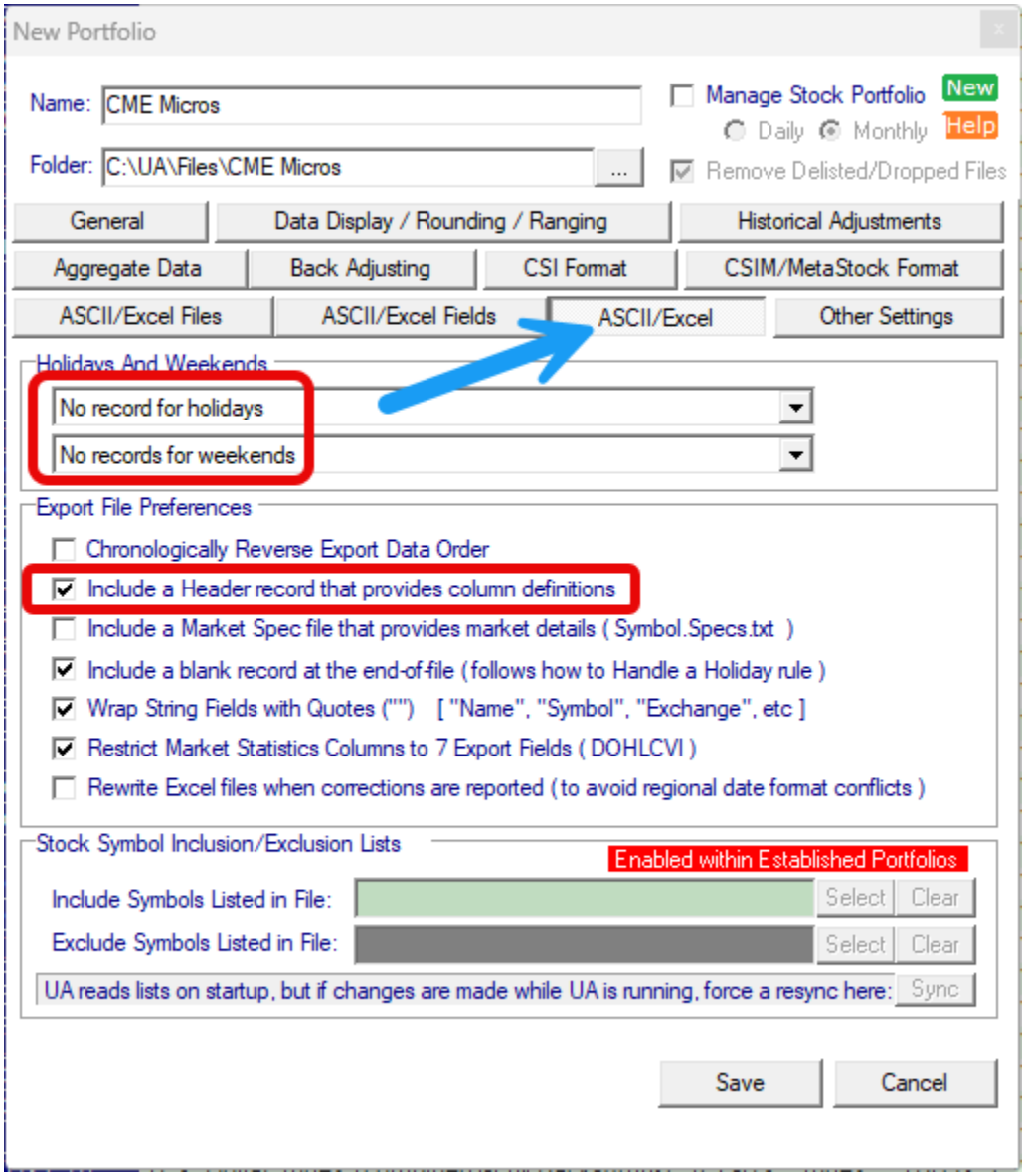
Expiry date is accessible via **InfoExpiry** or Event(-999), delivery number is accessible via Event(-998).

Expiry date is used to know when to add roll costs to a backtest and to specify the correct contract in CSV order lists.

Delivery number (year and month) is used to show the correct human-readable symbol (e.g. NG_2025F) in test logs and text order lists.

The above set of fields leaves one slot available in the RealTest bar records: **Volume**. You can use any of V, v, I, or i (total volume, contract volume, total OI, contract OI) as your volume field as desired, or leave it out if you don't need it. Whichever one of these is encountered first (left to right) will be used as Volume.

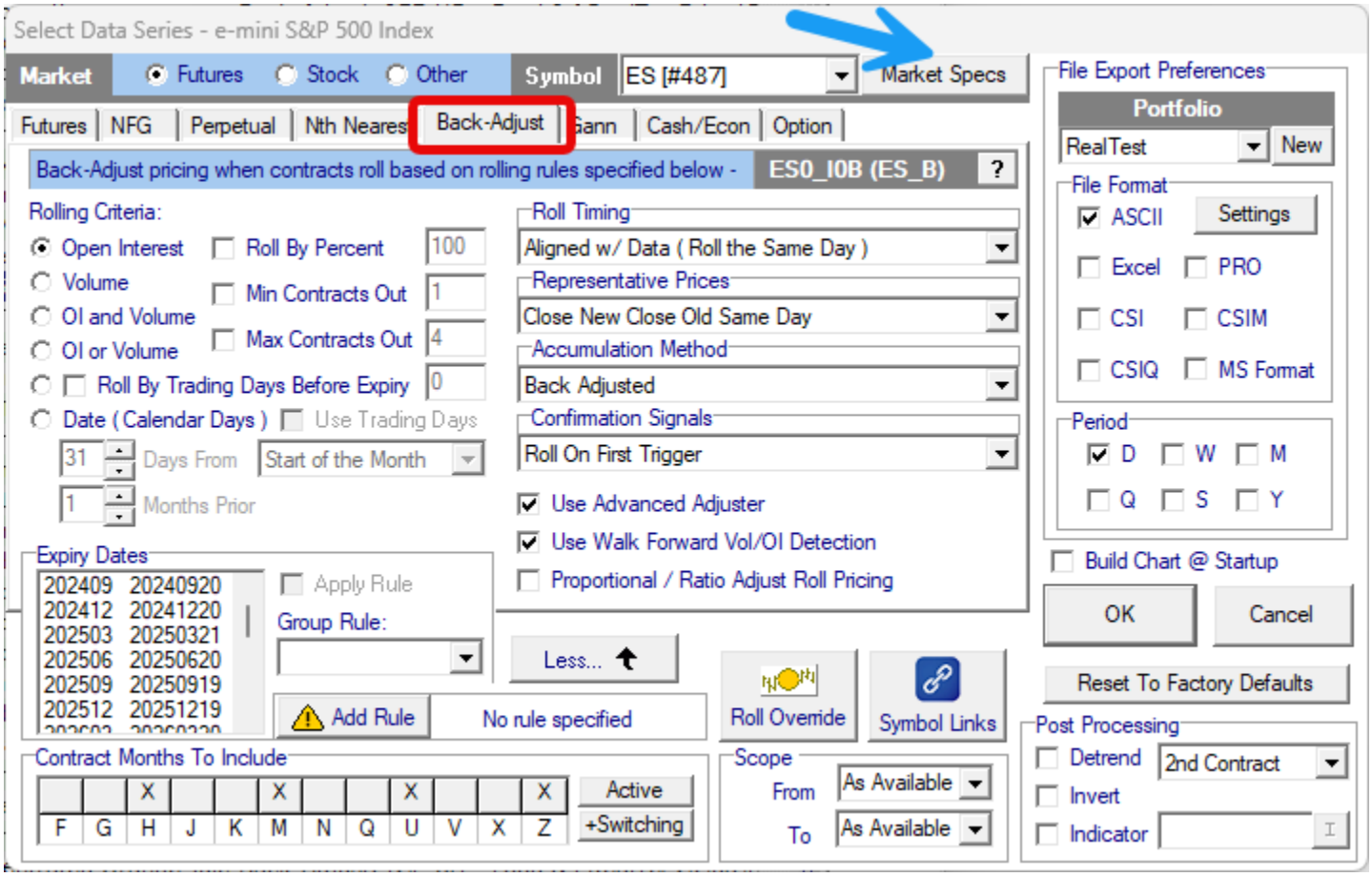
4. Specify the remaining details of your CSV files:



The "no record for holidays/weekends" settings are recommended but not required.

The "Include a Header Record" setting is required if you do not want to have to provide a **CSVFields** statement in your *Import* definitions.

5. Add symbols to your portfolio:



a. Click on "Add Symbol" in the main window

b. When this dialog appears, select the "Back-Adjust" tab

c. Configure your rolling criteria, timing, etc. (above is just an example -- do it however you prefer)

d. Type a symbol root or click on Market Specs to select one or more

e. Click OK

6. Once your portfolio is configured, UA will already have exported your CSV data files to your specified folder, and you can now import them into RealTest.

7. To keep your data up to date every day, simply run the "CSI EZ Download" program. It automatically downloads all the latest data to their internal database and then updates all of your portfolio files.

8. To import your CSI data into RealTest, the **Import** definition will look like this:

```
Import:
    DataSource:    CSIFutures
    DataPath:      c:\ua\Files\RealTest
    StartDate:     Earliest
    EndDate:       Latest
    SaveAs:        csi_test.rtd
```

You can add more than one **DataPath** if you want to combine data from multiple CSI portfolios into a single RealTest RTD file.

You can optionally name each *DataPath* using a comment like {"CME"} and later refer to it using e.g. **InList("CME")** if your formulas need to know which portfolio the current symbol came from.

As with CSV Import, you can optionally use **IncludeList** and/or **ExcludeList**. If you use *IncludeList*, this means "only include these symbols and exclude all others" (unless you also specify **KeepNonIncluded: True**). *ExcludeList* means "include all others except these".

Also as with CSV import, the file name becomes the symbol. Most often with CSI Futures, file names look like xx_b, where xx is the root symbol of the futures market and _b means "backadjusted". You will refer to those symbols as needed using e.g. Extern($cl_b, close).

Note that when you generate orders using this data, the symbols used in the orders represent the specific contract to be traded even when using the back-adjusted CSI data without individual contracts included.

Note also that with CSI Futures, you do not need to provide a **SymInfoFile**. RealTest reads and processes the **futuresinfo.txt** file found in each *DataPath* folder to get the market name, point value, and tick size for every symbol. This is why it's required to check that box in step (2) above.

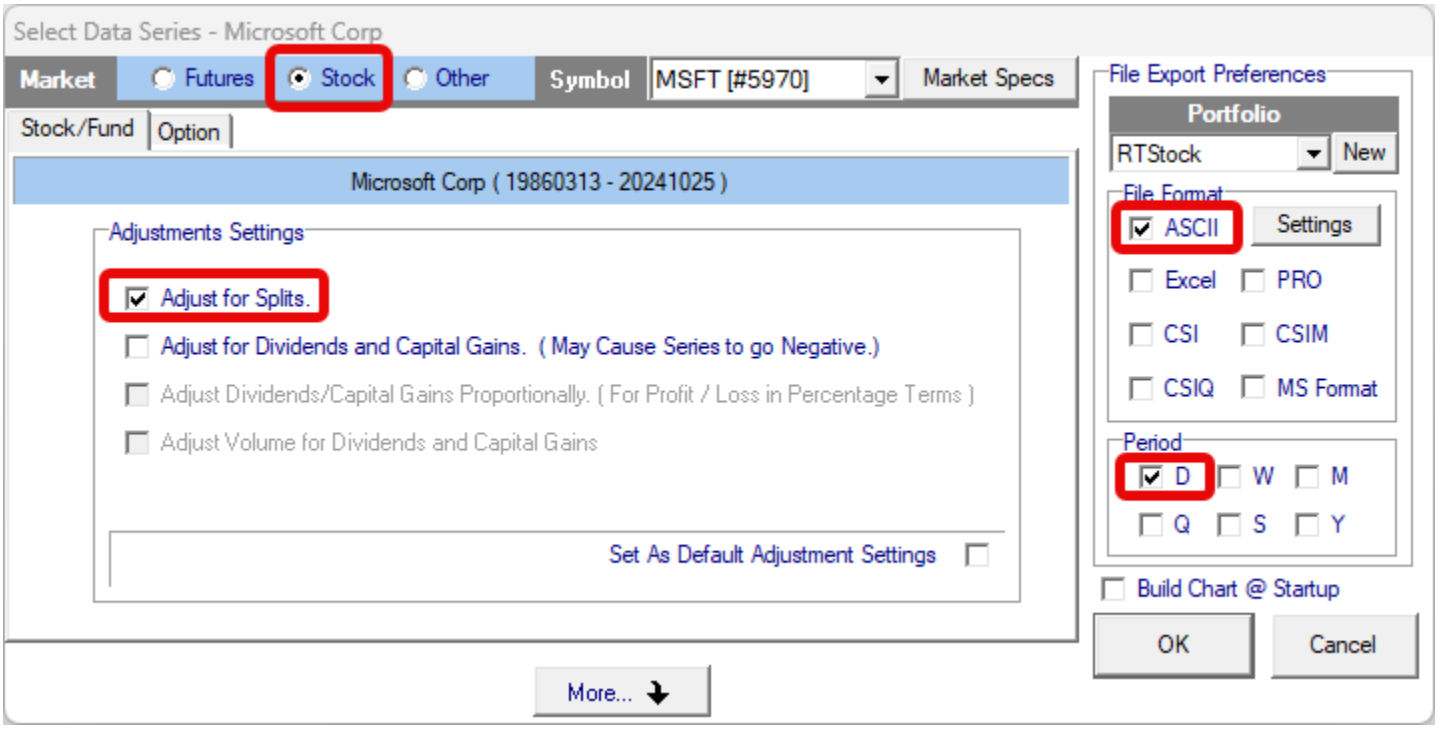
See **Using CSI Futures Data** for details about using this data in your backtests.

7.4. CSI Stocks Import

To use CSI stock data with RealTest, you must have downloaded and installed their **Unfair Advantage** software and subscribed to one of their stock data packages.

Within the *Unfair Advantage* program, create one or more new stock portfolios for use with RealTest.

Stock portfolios for use with RealTest must be configured as follows:



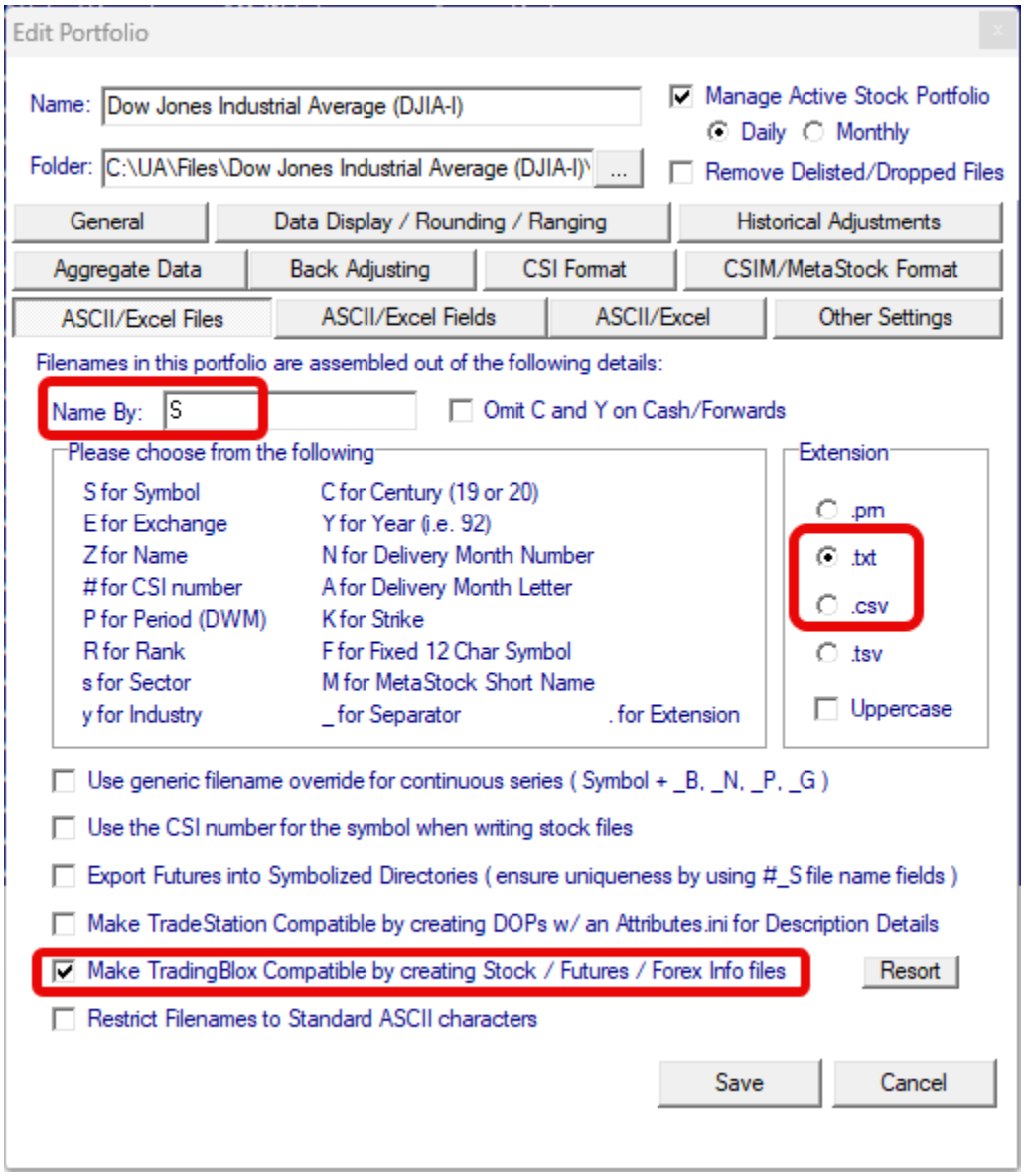
The items in red are required. Adjustment for dividends is optional. If your subscription includes dividend details it is preferable to *not* use dividend adjustment.

Note that the **Adjustment** setting of the **Import** section is not supported with the *CSIStocks* data source.

The UI settings as shown above correspond to *Adjustment: Capital* (the default) in RealTest.

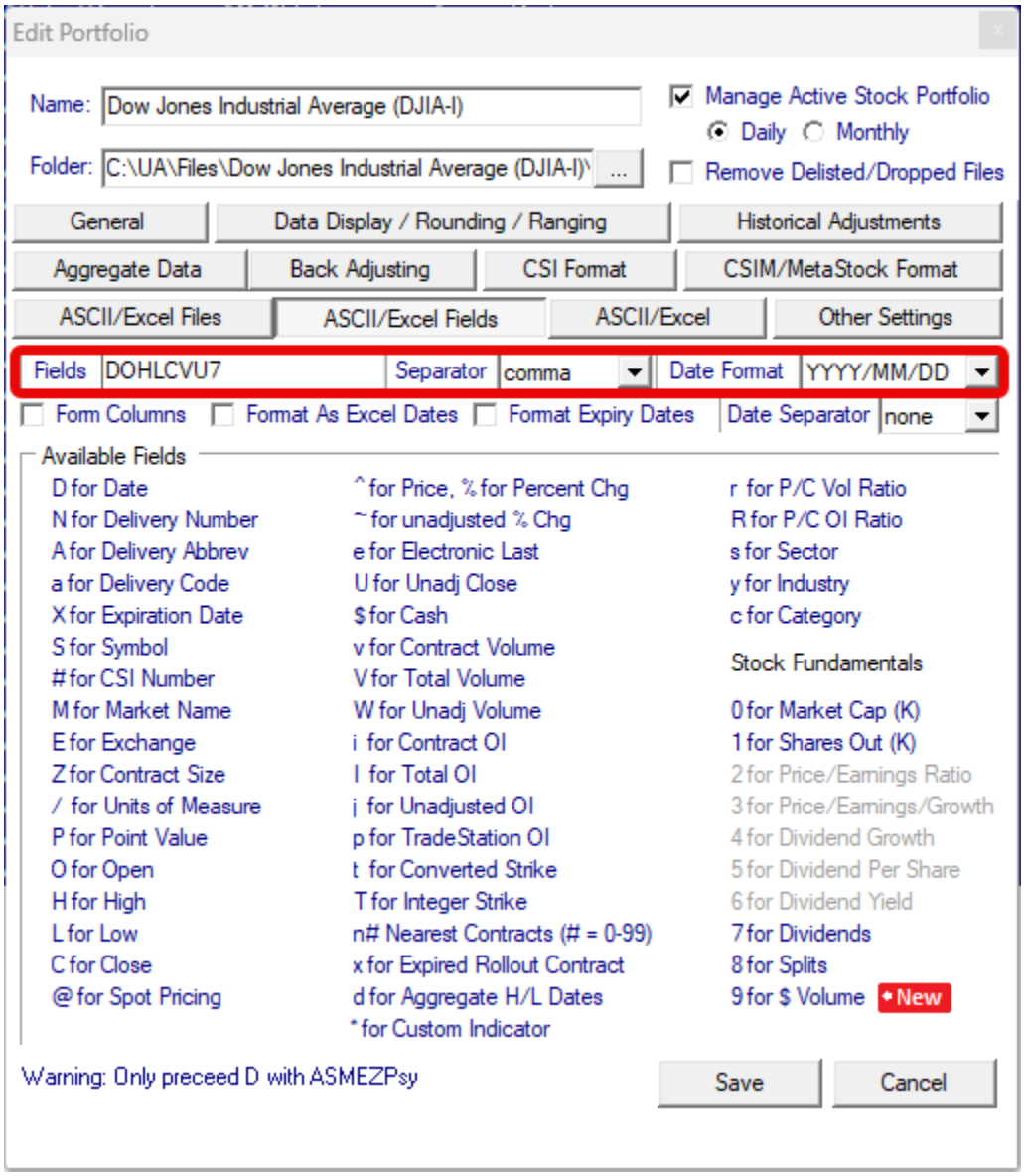
If you want your imported data to use *TotalReturn* adjustment (in which dividends are converted to splits), then you must also check the other three boxes under "Adjust for Splits" in the above dialog box and rebuild your UA portfolio.

Use these settings to further configure your portfolio:



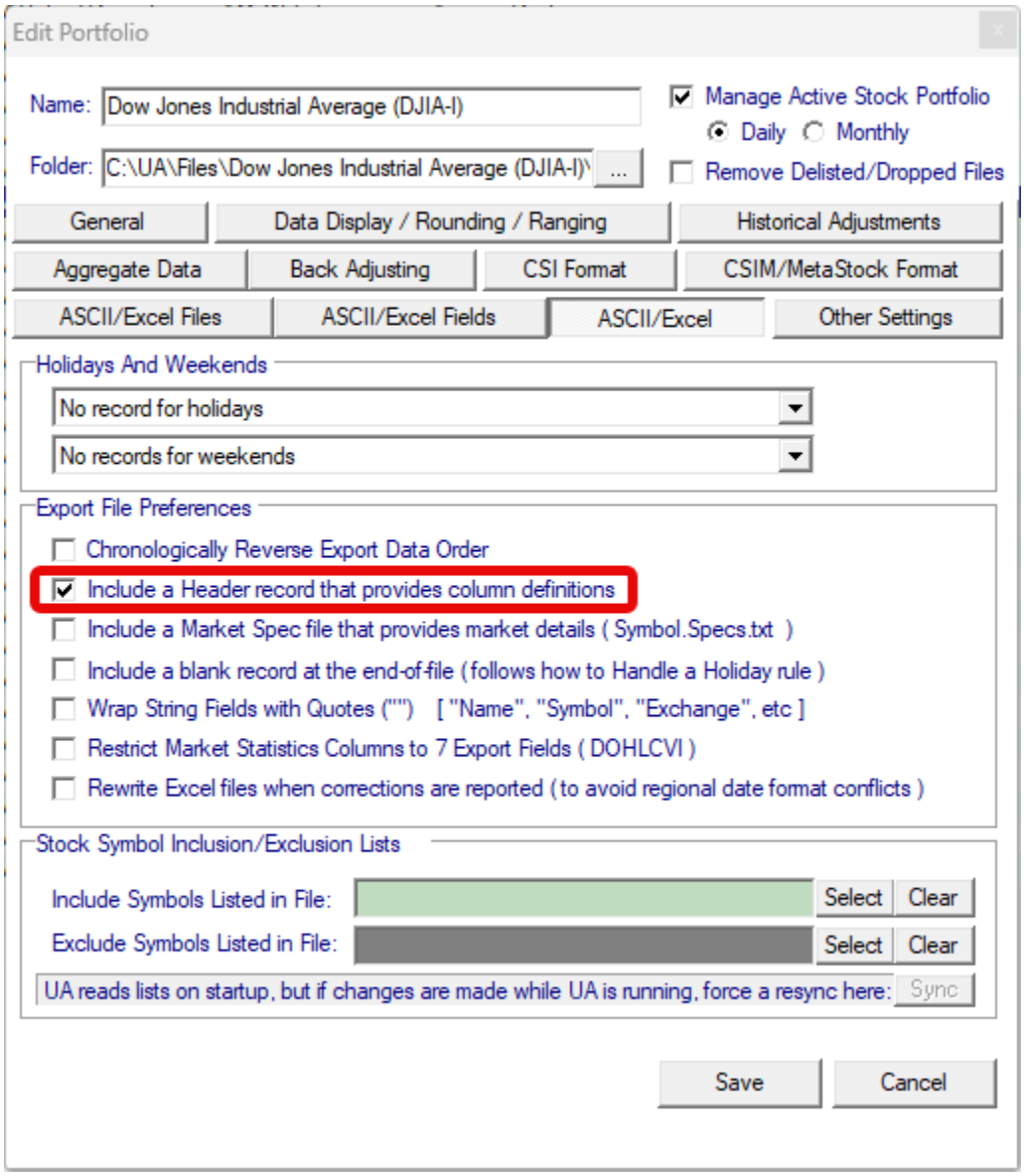
Name By: S is required (just the symbol, nothing else in the file name).

Extension can be either .txt or .csv -- your choice.



Fields should include all of the above. The sequence does not matter.

Separator must be comma and date format as shown.



Header row is required.

Once your portfolio is configured, UA will already have exported your ASCII data files to your specified folder, and you can now import them into RealTest.

To keep your data up to date every day, simply run the "CSI EZ Download" program. It automatically downloads all the latest data to their internal database and then updates all of your portfolio files.

To import your CSI stocks data into RealTest, the **Import** definition will look like this:

```
Import:
    DataSource:    CSIFutures
    DataPath:      c:\ua\Files\RealTest
    StartDate:     Earliest
    EndDate:       Latest
    SaveAs:        csi_test.rtd
```

You can add more than one **DataPath** if you want to combine data from multiple CSI portfolios into a single RealTest RTD file.

You can optionally name each *DataPath* using a comment like {"DJI"} and later refer to it using e.g. **InList("DJI")** if your formulas need to know which portfolio the current symbol came from.

As with CSV Import, you can optionally use **IncludeList** and/or **ExcludeList**. If you use *IncludeList*, this means "only include these symbols and exclude all others" (unless you also specify **KeepNonIncluded: True**). *ExcludeList* means "include all others except these".

Note also that with CSI Stocks, you do not need to provide a **SymInfoFile**. RealTest reads and processes the **stockinfo.csv** file found in each *DataPath* folder to get the company name etc.

7.5. 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.

Since RealTest symbol names cannot contain spaces, any symbol name containing a space, tab, parenthesis, or comma is normalized on import: each such character becomes an underscore, and consecutive underscores are collapsed to one. This applies to all local-file data sources -- see **MetaStock Import** for details.

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 (auto-detected from the header row, or specified with CSVFields)
- 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 can include a **CSVFields** statement to define the column layout; if omitted, RealTest auto-detects the layout from the header row. A CSVFields mapping carries forward to later sources until another one replaces it; use CSVFields: Auto to reset a later source to header auto-detection

Multi-file CSV import (one file per symbol via **DataPath**) is multi-threaded, so import speed for large file collections scales with the number of available CPU cores.

See also **import_csv.rts** in the **Examples** folder for an example.

7.6. EODHD Import

The **EODHD** data service is a significant step up from Yahoo in terms of quality and reliability.

EODHD.com offers a free service level with severe usage limitations, or virtually unlimited usage for $20/month.

EODHD includes a broad range of countries among their data offerings.

Whether you use their free or paid service, you must register with them to receive a private API key.

Once you have your API key, enter it on the Import tab of the **Program Options** dialog, in the **EODHD API key** field (use the **EODHD exchange** field to set a default country code other than US).

Alternatively, you can set these by hand in RealTest.ini:

1. Close RealTest if it is running.
2. Find your **RealTest.ini** file (it will be in c:\RealTest or wherever you installed RealTest) and open it using a plain text editor such as Notepad.
3. Search for **Eodhd** to find the line labeled**EodhdApiKey**.
4. Copy and paste your API key after the equals sign.
5. If you'll primarily be importing data from a country other than the US, change **EodhdExchange** to the country code (which they call "exchange") of that country.
6. Save RealTest.ini, close Notepad, run RealTest.

You are now able to import from EODHD. Your key will remain preserved in your RealTest.ini file thereafter unless you delete or replace the file or edit it again and delete the key.

Note that regardless of your EodhdExchange (country code) setting, you can import symbols from any country by writing them with the correct country code as a suffix. For example, MSFT could also be imported as MSFT.us, the Australian stock BHP can be imported as BHP.au, etc. no matter what your EodhdExchange setting is. The purpose of the setting is to allow you to select one country that will not require an exchange suffix on all of its symbols.

7.7. 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. In each folder, RealTest reads the EMASTER (or XMASTER) index file when present, otherwise MASTER.

Symbol Names

Since RealTest symbol names cannot contain spaces, vendor symbol names are normalized on import: each space, tab, parenthesis, or comma becomes an underscore, and consecutive underscores are collapsed to one. For example, a symbol stored as *Nifty 50* imports as *Nifty_50* and can be referenced in formulas as *Extern($Nifty_50, C)*. All other characters, and letter case, are preserved. If two names normalize to the same symbol, the first is kept and later duplicates are skipped.

This normalization applies to symbol names from any local-file data source, not just MetaStock.

See also **import_ms.rts** in the **Examples** folder for an example.

7.8. Tiingo Import

The **Tiingo** data service is a significant step up from Yahoo in terms of quality and reliability.

Tiingo offers a free service level with usage limitations, or virtually unlimited usage for $30/month.

Whether you use their free or paid service, you must register with them to receive a private API key.

Once you have your API key, enter it on the Import tab of the **Program Options** dialog, in the **Tiingo API key** field.

Alternatively, you can set the key by hand in RealTest.ini:

1. Close RealTest if it is running.
2. Find your **RealTest.ini** file (it will be in c:\RealTest or wherever you installed RealTest) and open it using a plain text editor such as Notepad.
3. Search for **Tiingo** to find the line labeled**TiingoApiKey**.
4. Copy and paste your API key after the equals sign.
5. Save RealTest.ini, close Notepad, run RealTest.

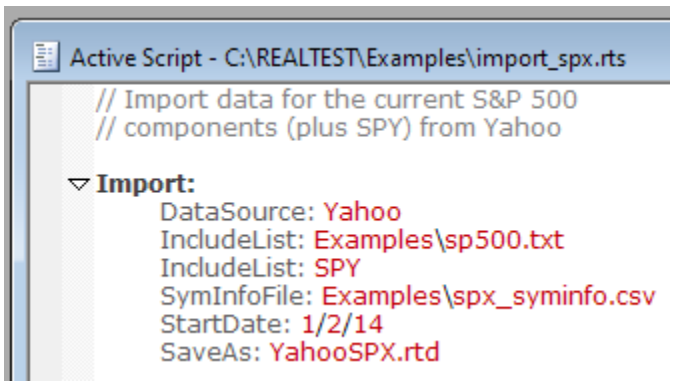
You are now able to import from Tiingo. Your key will remain preserved in your RealTest.ini file thereafter unless you delete or replace the file or edit it again and delete the key

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 cryptocurrency symbol.

The **Tiingo exchanges** field on the Import tab (or the *TiingoExchanges* line in RealTest.ini) can optionally be used to provide the &exchanges string to send to the Tiingo API when requesting data.

7.9. 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.



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.

7.10. 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:\NDExport1 // include all symbols
    DataPath: c:\NDExport2 // ditto

    // Local CSV directories where we want only a subet of the symbols
    DataSource: CSV // required to prevent IncludeList below from applying to DataPaths above
    IncludeList: AAPL,VZ
    DataPath: c:\NDExport3
    DataPath: c:\NDExport4

    // 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:

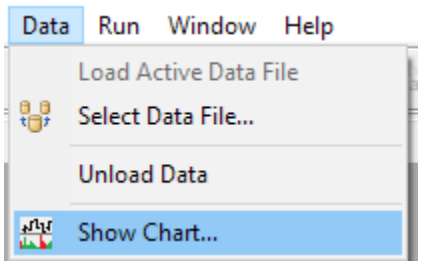
- &ES (E-Mini S&P 500 futures continuous contract) and SPY would be imported from Norgate
- SPY would also be imported from Yahoo with its symbol renamed to "SPY2"
- AAPL would be imported from a CSV data directory (assuming there is a file named AAPL.CSV in that directory)
- All symbols in a second 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

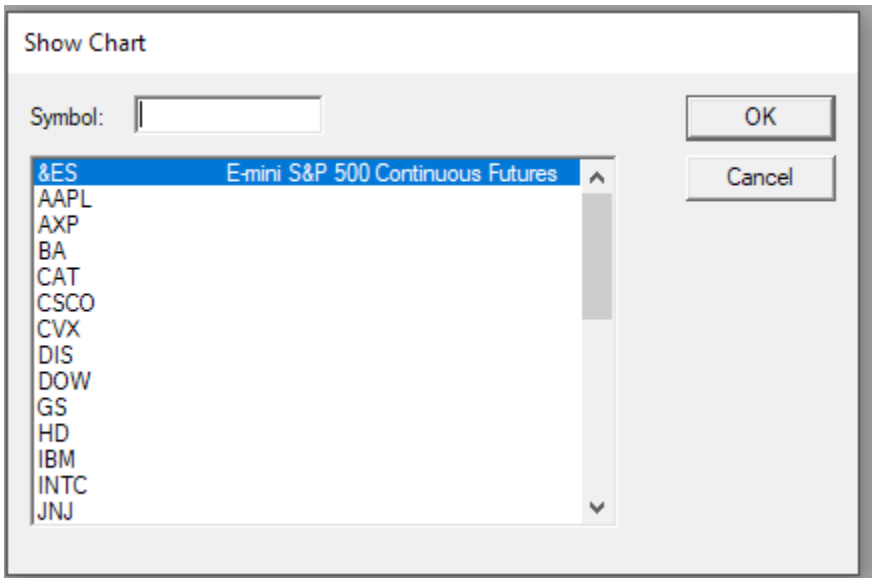
The log file confirms that all the symbols were imported:

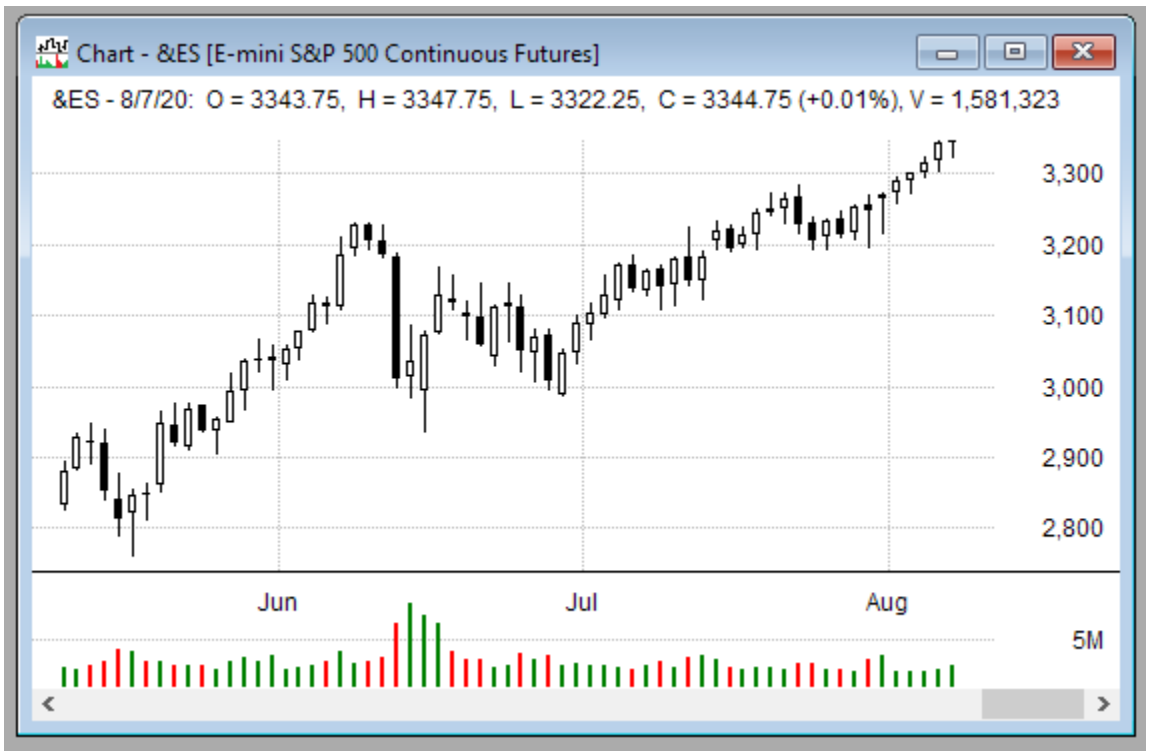
Log - c:\realtest\multi.log*

```
-------------------------------------------------------------
Importing Norgate Data From NDU
-------------------------------------------------------------
Symbol          FirstDate       LastDate        Bars            Events
&ES             1/2/18          8/7/20          655             0
SPY             1/2/18          8/7/20          655             10
-------------------------------------------------------------
Importing Yahoo Data From finance.yahoo.com
-------------------------------------------------------------
Symbol          FirstDate       LastDate        Bars            Events
SPY2            1/2/18          8/7/20          655             10
-------------------------------------------------------------
Importing CSV Data From c:\NDExport
-------------------------------------------------------------
Symbol          FirstDate       LastDate        Bars            Events
AAPL            8/14/18         8/7/20          500             0
-------------------------------------------------------------
Importing CSV Data From c:\NDExport2
-------------------------------------------------------------
Symbol          FirstDate       LastDate        Bars            Events
AXP             8/14/18         8/7/20          500             0
BA              8/14/18         8/7/20          500             0
CAT             8/14/18         8/7/20          500             0
CSCO            8/14/18         8/7/20          500             0
CVX             8/14/18         8/7/20          500             0
DIS             8/14/18         8/7/20          500             0
DOW             3/20/19         8/7/20          351             0
GS              8/14/18         8/7/20          500             0
HD              8/14/18         8/7/20          500             0
IBM             8/14/18         8/7/20          500             0
INTC            8/14/18         8/7/20          500             0
JNJ             8/14/18         8/7/20          500             0
JPM             8/14/18         8/7/20          500             0
KO              8/14/18         8/7/20          500             0
MCD             8/14/18         8/7/20          500             0
MMM             8/14/18         8/7/20          500             0
MRK             8/14/18         8/7/20          500             0
MSFT            8/14/18         8/7/20          500             0
NKE             8/14/18         8/7/20          500             0
PFE             8/14/18         8/7/20          500             0
PG              8/14/18         8/7/20          500             0
RTX             8/14/18         8/7/20          500             0
TRV             8/14/18         8/7/20          500             0
UNH             8/14/18         8/7/20          500             0
V               8/14/18         8/7/20          500             0
VZ              8/14/18         8/7/20          500             0
WBA             8/14/18         8/7/20          500             0
WMT             8/14/18         8/7/20          500             0
XOM             8/14/18         8/7/20          500             0

Saved As c:\realtest\multi.rtd

Import Was Completed
```

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







7.11. Symbol Information Files

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 Variable
Symbol	string	**?Symbol**

Name	string	**?Name**
Type	string	**?Type**
EquityType	string	**?EquityType**
ListingType	string	**?ListingType**
Exchange	string	**?Exchange**
Currency	string	**?Currency**
ReportingCurrency	string	**?ReportingCurrency**
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**
Margin	numeric	**InfoMargin**
PointValue	numeric	**PointValue**
TickSize	numeric	**TickSize**
AssetId	numeric	**InfoId**
DelistDate	date	**InfoDelist / InfoExpiry**
PriceMult	number	n/a

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

The set of supported symbol information columns corresponds to the metadata that is automatically included when **Norgate** is the data source. There is therefore rarely a reason to use a **SymInfoFile** with Norgate import.

The one exception to the above is PriceMult. This does not correspond to a script variable. Rather, it is used to enable automatic correction during import when data services multiply prices by (typically) 100, most often for JPY futures. The open, high, low and close are divided by the PriceMult value, as are the dividend amount and the adjusted or unadjusted close column when the file includes them, so that everything stays in the same units.

PriceMult is not needed for London stocks quoted in pence. A Currency of GBp (Yahoo's spelling for pence-quoted lines, which you can also specify in a symbol information file for any data source) is handled automatically: prices stay in pence and an internal point value of 0.01 converts them to pounds wherever values are computed. To import such a stock in pounds instead, set its Currency to GBP with a PriceMult of 100; the point value then defaults to 1.

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.

The *Type* column determines how a symbol is modeled in a backtest. A value beginning with *Fut* is a futures contract, *Forex* or *FX* is a currency pair, *Crypto* is a 24/7 symbol, and anything else (including no Type at all) is a stock. The distinction matters most when a **Currency** setting is in effect: a stock's profit includes the exchange-rate change on its full value between entry and exit, while a futures or forex position's profit is the points gained times the quantity, converted to the account currency at the exit rate (see **Testing Multi-Currency Strategies**). Currency pairs imported from a source other than Norgate therefore need *Type* set to *Forex* along with their quote *Currency*, and should also be given a *TickSize* (e.g. 0.00001), because the default tick size of 0.01 rounds fills of non-stock symbols to two decimal places.

Your Import definition can include any number of SymInfoFile statements. For data services like Norgate that already provide metadata, any values you provide will override the corresponding defaults. Each SymInfoFile overrides any values specified by prior ones in the script's line sequence.

See also **import_yahoo.rts, spx_syminfo.csv, djia_make_syminfo.rts, djia_use_syminfo.rts**, and **djia_info.csv** in the **Examples** folder.

7.12. Futures Symbol Information

For futures data import from a source other than **Norgate** or **CSI Futures**, it is critical to include a **SymInfo file** with the correct **PointValue** for every symbol. Otherwise, backtest results will not be accurate. (**TickSize** is not required, but if present, it will be used in chart scaling, and will be useful in formulas such as slippage modeling.)

A futures SymInfo file must also include a "Type" column containing anything starting with "Fut" in each row. This is how RealTest knows the symbol is a futures contract.

You can import data that contains any mix of asset types. RealTest models transactions for each symbol according to its asset type. This is why it's important to tell RealTest what type a symbol is.

The *Margin* column in the *SymInfo file*, if provided (it's not required), specifies the current dollars-per-contract margin collateral for each futures symbol. By default RealTest uses the current Margin value for all historical dates in a backtest. You can optionally provide a formula to calculate estimated historical margin requirements via the **Collateral** strategy element.

RealTest does not automatically limit futures exposure based on margin requirements but can optionally do so using the **MinFreeCash** constraint formula and/or the **S.Collateral** stats array. The **S.FreeCash** stats array (referenced internally for *MinFreeCash* determination) assumes that a position requires futures margin only if the symbol has a non-zero margin requirement.

7.13. Event List Files

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 symbol information files, event lists use CSV format.

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 – a numeric code > 0, or an event type name (any text that is not a number); names are matched without regard to case, and up to 64 distinct names may be used
- 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 at least one event for one symbol on one date. Additional *Type,Value* pairs may be appended to a row after its header-named columns; each pair defines another event that shares the same Symbol, Date and Time (up to 64 events per row). Give each pair on a row a distinct type, since only one event per type is retrievable for a given symbol and date.

For example, this file defines one event for AAPL and three (two of them with named types) for MSFT:

```
Symbol,Date,Time,Type,Value
AAPL,2026-01-30,163000,1,2.18
MSFT,2026-01-28,163000,1,3.05,EarnEst,2.90,Surprise,0.15
```

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 reported 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). An event type can be referenced either by its number, e.g. *Event(1)*, or by its quoted name, e.g. *Event("EarnEst")*.

The *Time* field stored with each event record determines which bar the Event function will align the event with. If Time <= 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.

Events dated *after* the last bar of imported data are retained. This is what allows **UntilEvent** and negative-offset references like *Event(n)[-5]* to see future-dated events during order generation. (A symbol's most recent imported bar must still exist on or before the event date for it to remain associated with that symbol.)

See also **djia_earnings.rts** and **djia_earnings.csv** in the **Examples** folder (note that this example does not use the *Time* field).

7.14. Dividend and Constituency Events

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 in **Event List Files**, RealTest also uses the event list within each RTD file to store **Dividends** and **Constituency Change** 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 as 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!

8. Running Scans

A scan outputs time-series data in a customizable table. These topics describe how scans work and each type.

- **How Scans Work** - the scan types and how each is run
- **Data Scan** - scanning OHLCV and Data-section arrays outside a backtest
- **Test Output Scan** - scanning with strategy and test context during a backtest
- **Multi-Filter Scan** - producing multiple output rows per symbol or date

8.1. How Scans Work

Scans in RealTest allow you to display time-series data in tables with easily customizable content.

RealTest supports several types of scans:

- **Data Scan** - has access to OHLCV data and calculated **Data Section** arrays, runs separately from backtests
- **Test Scan** - same as Data Scan but runs during backtests and can therefore also access strategy and test information
- **Test Scan Positions** - same as Test Scan but loops over current positions rather than all symbols
- **Multi-Filter Scan** - all scan types support output of multiple rows per scan unit (symbol or position) by defining multiple filter formulas

To run any type of scan, you must first import or load a data file, then open or create a script that includes either a **Scan** section or one or more **Strategy** sections plus a **TestScan** section.

A Data Scan is run by pressing the "Scan" button in the toolbar.

A Test Scan is run by selecting "Scan" under Test Output and then pressing "Test" or "Orders".

The other topics in this section provide more details about each type of scan.

8.2. Data Scan

This is the most typical type of scan and is simply called "Scan" in RealTest.

See **Tutorial 3** for a complete example of a data scan.

To run a data 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 specified (overriding any panel settings), then applies the **Scan Settings** section if specified (overriding panel and script settings). The scan is then run by looping over all symbols and all dates using the **BarSize** setting.

If the Scan section includes a *Filter* formula, that is evaluated first for each bar of each symbol. If the Filter cannot be evaluated (nan) or returns False (0) then that bar is excluded from the scan output and the remaining formulas are not evaluated for it.

All other formulas defined in the Scan section are then evaluated and their results are added to the output.

The default output of a Data Scan is a **Scan Window** containing a row for every bar that was not filtered and a column for every named formula in your Scan section definition, preceded by the standard Date and Symbol columns.

Scan definitions can optionally include a *Sort* item to specify the initial sort order of the rows. This special item is not a formula like all other Scan items. Rather, it is a comma-separated list of column names. Precede a name with a minus sign to sort by that column in descending order.

The following choices can optionally be specified in Settings or ScanSettings to change the default behavior of scans:

- ScanNoDefCols - suppress the output of the Date and Symbol columns
- SaveScanAs - write the output to a CSV file as well as displaying it in a window
- ScanNoWindow - do not display the scan in a window (requires SaveScanAs)
- ScanNoHeader - omit the column name header row from the SaveScanAs CSV output file

8.3. Test Output Scan

A Test Output Scan (TestScan) is similar to a **Data Scan** (Scan).

To run a TestScan, the script that defines it must also include one or more Strategy and/or Benchmark definitions, must specify **TestOutput: Scan** (or that option must be selected in the **Settings Panel**), and must be run in Test or Orders mode.

The structure of the Scan and TestScan definitions in the script look identical. The differences come from when they are run and what syntax their formulas have access to.

Because a Data Scan is not run in the context of a backtest, none of the syntax elements that relate to backtests -- position information, test statistics, etc. -- are available for use in Scan formulas. TestScan formulas have full access to that additional set of information.

By default, a TestScan is run only once per test, at the end of the test. Specifically, it is run after the strategy formulas have been evaluated and applied on the last date of the test but before the end-of-test exits have been processed and added to the trade list. To run your TestScan every day of the test, add **TestScanAllDates: True** to your script's **Settings** or **TestSettings** section.

Because a script can include multiple Strategy definitions but has only one TestScan, any reference to anything about a strategy must be done using either the **Combined** or **Extern** functions.

For example, to show the net position in a symbol held across all strategies, use:

```
TestScan:
   NetPos: Combined(Shares)
```

Or to see the position held by a specific strategy:

```
TestScan:
   Strat1Pos: Extern(@Strat1, Shares)
```

By default, for each date that the TestScan is run, it evaluates each formula once per symbol, much like a DataScan does. To run your TestScan for each open position rather than each symbol, add **TestScanPositions: True** to your Settings or TestSettings. When *TestScanPositions: True* is used, an output row is added separately for each specific position, including multiple positions in the same symbol. In this mode, there is no output for symbols that do not have an open position on that date of the test.

The Filter, Sort, and special scan-related settings described for **Data Scan** are similarly applicable to TestScan output.

If a script includes both Scan and TestScan sections and needs different scan-related settings for each, use the **ScanSettings** and **TestSettings** sections to differentiate them.

8.4. Multi-Filter Scan

By default, RealTest scans will contain, at most, one row per symbol per date, or one row per position per date for TestScan in *TestScanPositions: True* mode.

It is also possible to output multiple rows for the same symbol or position on the same date, with different contents.

Before RealTest provided a built-in order generation feature, scans were used to create CSV order lists e.g. for IB Basket Trader input. This was the original use case for multi-filter scans.

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. Each row is only output for any filter if its formula returns a non-zero value ("TRUE") for that symbol on that date.

To add multiple filters to a scan definition, start the name of each one with the word "filter" and then add one or more other characters to make their names unique. RealTest interprets any Scan or TestScan formula name that begins with "filter" as a filter definition rather than a column definition.

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)
```

Scan - 20 Items

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

This shows how the Scan includes two filters with the same condition. This simply produces two rows per symbol per day for each symbol that meets the specified "setup" condition. The Action and LimitPrice formulas use FilterNum to produce different output for each row for the same symbol.

9. Running Tests

This section covers running backtests, from a single test to many at once, including walk-forward models.

- **How Tests Work** - launching a test and how settings are applied
- **Multiple Tests and Optimization** - running many tests and optimizing parameters
- **Walk-Forward Tests** - simulating periodic re-optimization

9.1. How Tests Work

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**.

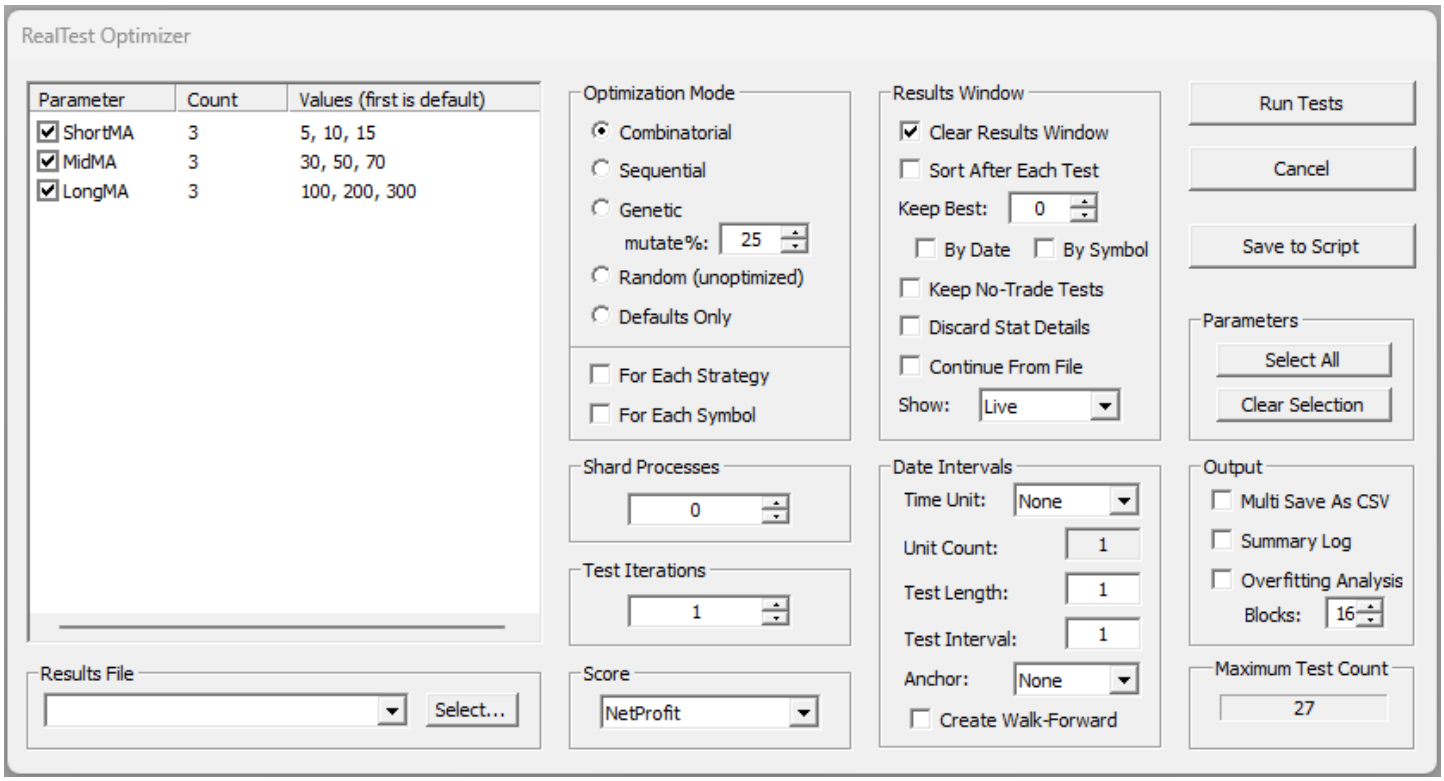
9.2. Multiple Tests and Optimization

Optimize mode is a special run mode that performs multiple back-to-back tests. Despite the name, it is not only for parameter optimization - it is the right mode for any task that requires more than a single test, including:

- exploring a parameter space (the traditional optimization use case)
- running the same script across multiple date ranges
- generating a **walk-forward** test
- repeating a randomized test many times to study the distribution of outcomes
- splitting a run by strategy or by symbol

See **Tutorial 2 - Optimization** for a specific usage example.

To start Optimize mode, click **Optimize** in the toolbar or choose **Run / Run Optimization** from the menu. The RealTest Optimizer dialog appears before the test starts, giving you a chance to configure the run:



The sections below describe each area of the dialog. All controls remain enabled at all times; rather than silently adjusting settings, the dialog validates the combination you have chosen when you start the run and reports any conflict.

Parameter List

The left panel lists every entry in the script's **Parameters** section, along with the number of values defined and a preview of the value list. The checkbox next to each parameter controls whether that parameter is varied during the run - unchecked parameters always use their default (first) value.

By default no boxes are checked initially. A parameter definition whose comment includes the special tag *{X}* is checked by default.

The **Select All** and **Clear Selection** buttons toggle every box at once.

Optimization Mode

This group chooses how the parameter space is searched:

- **Combinatorial** is the exhaustive nested loop - every combination of the checked parameter values is tested. Test counts grow quickly with the number of varied parameters, but RealTest can handle very large runs - portfolio-level optimizations involving thousands of tests across thousands of stocks over many years are quite practical. Whether this approach is likely to discover a strategy that will be profitable in the future is another matter.
- **Sequential** sweeps one parameter at a time. At the end of each sweep, the value that produced the highest score is locked in and the next parameter is swept. Two or three full iterations of a sequential run will often find a good neighborhood faster than the combinatorial approach.
- **Genetic** is a randomized search inspired by genetic optimization, but not a true genetic optimizer. Before each test, a random subset of parameters is set to new random values from their lists. (How many parameters change each step is controlled by the **mutate%** setting.) If the score improves, the change is kept; otherwise the prior best values are restored. Genetic mode typically converges on the best combinatorial result within roughly the square root of the total combination count. Requires Test Iterations.
- **Random** is similar to Genetic but every parameter is rerolled before every test and the score is ignored. A useful application is to set reasonable value ranges for each parameter, run 100 iterations, and look at the median result - this can give a reasonable estimate of how the system might perform in the future given how arbitrary parameter selection often is. Requires Test Iterations.
- **Defaults Only** is equivalent to unchecking every parameter - the script runs once using each parameter's

default value. Combined with **For Each Strategy**, **For Each Symbol**, or **Date Intervals**, this is how to run a multi-test job with no parameter variation.

For Each Strategy / For Each Symbol

These checkboxes split the job into one test (or one whole optimization) per strategy or per symbol, as if every other strategy were commented out or every other symbol were absent from the data. They can be combined - doing so produces strategy × symbol tests.

Shard Processes

Specifying a count > 1 enables parallel sharded optimization - see **Parallel Optimization (Shards)** below.

Test Iterations

The number of times to run the configured test. Use this to:

- repeat a test that uses the **Random** function in one or more formulas, to obtain a range of potential outcomes
- specify how many tests Genetic mode should run
- specify how many tests Random mode should run

Score

The Results column to use as the optimization's fitness function. Because Results columns are formula-based, you can score by any expression - net profit, Sharpe, MAR, or any custom formula you have defined. The Score column is used to rank rows when **Sort After Each Test** is checked, to drive the Sequential and Genetic searches, and to select best parameter sets when building a Walk-Forward test.

Results File

The field in the lower-left corner sets the path and name of the RTR results file to create or open for this run - the dialog equivalent of a **ResultsFile** setting in the script's **OptimizeSettings** section. Use the **Select...** button to browse for a file. Setting it here means you don't have to cancel, add a ResultsFile setting to the script, and reopen the dialog. A Results File is required by **Continue From File** and by the **AtEnd** and **None** Show modes.

Results Window

Controls how the Results window and the underlying RTR file are populated during the run.

- **Clear Results Window** - empty the window before the run starts (otherwise new test rows are appended to the existing list).
- **Sort After Each Test** - re-sort the rows by Score after each test completes. Pair this with Keep Best to retain only the top rows as the run progresses.
- **Keep Best** - keep only the N highest-scoring rows during the run. **By Date** applies the limit to each Date Interval separately, and **By Symbol** applies it to each For Each Symbol test. Requires Sort After Each Test.
- **Keep No-Trade Tests** - by default a parameter combination that produces zero trades is discarded. Check this to keep those rows.
- **Discard Stat Details** - drop the per-period statistics array from each saved test, keeping only the precomputed Results column values. This dramatically reduces RTR size and memory use, making optimizations of millions of tests feasible. Rows saved this way can no longer drive the Graph, Plot, Report, Save Stats, or Save Trades menu items - only their Results column values are available.
- **Continue From File** - pick up where a previously interrupted optimization left off. RealTest

seeds the run from the parameter combinations already in the **ResultsFile** RTR and skips those combinations during enumeration. Requires Combinatorial mode, a ResultsFile setting, Test Iterations = 1, no Keep Best pruning, no Date Intervals, no For Each Strategy/Symbol, and no Walk-Forward. Optimizations that have a ResultsFile setting now save their RTR file frequently during the run, so a crash, power loss, or manual stop loses at most a few minutes of completed tests.

- **Show** - chooses when the Results window is opened. **Live** updates the window as each test finishes (the historical behavior). **AtEnd** runs the optimization headless and opens the RTR file when the run completes. **None** runs headless and never opens the file - useful for batch pipelines where the RTR is the final artifact. **AtEnd** and **None** require **ResultsFile** and skip the per-test list update, which materially speeds up very large optimizations.

Date Intervals

The Date Intervals panel is used primarily to generate a **walk-forward** test, but can also 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.) Setting the time unit to "Years" is a convenient way to produce one result row per calendar year of a date range.

By default each date interval is a sliding window of the same length. The **Anchor** setting optionally locks the start or end date. Anchoring the start date makes each interval longer than the preceding one; anchoring the end date does the opposite.

Create Walk-Forward

Generates a walk-forward test from the configured Date Intervals and parameter selection. See **Walk-Forward Tests** for a detailed description of this feature.

Output

- **Multi Save As CSV** - perform **SaveStatsAs** and/or **SaveTradesAs** once per test, creating separate files for each. These settings are otherwise ignored in a multi-test run.
- **Summary Log** - write a log window summarizing the optimization when the run finishes.
- **Overfitting Analysis** - analyze the completed optimization for overfitting: how often the in-sample best parameter combination held up out-of-sample (CSCV/PBO), and how well the whole parameter surface carried from in-sample to out-of-sample (walk-forward correlation). **Blocks** sets how many time blocks the test period is cut into (even, 4 to 20). See **Overfitting Analysis** for a full description. Cannot be combined with Walk-Forward, Date Intervals, For Each Strategy / Symbol, Shard Processes, or Continue From File.

Maximum Test Count

The number in the lower-right corner of the dialog shows how many tests the current settings will produce. Watch it change as you toggle checkboxes and adjust options - it is the quickest way to see whether the run will take seconds, minutes, or days before committing to it.

Parallel Optimization (Shards)

For very large Combinatorial runs, the **Shard Processes** setting divides the parameter space across multiple parallel RealTest processes (2 to 64). Each shard runs an independent slice of the largest active parameter's value list and writes its own RTR file. When all shards complete, RealTest merges them into a single combined RTR.

Sharding has the following requirements:

- Combinatorial, Genetic or Random mode (sharding is not supported for Sequential mode).
- No Date Intervals, For Each Strategy, For Each Symbol, Walk-Forward.

- More total tests than shards.

If the dialog settings end up with nothing meaningful to slice across, the optimization runs inline in a single process instead, so *OptShards* is safe to leave set in scripts that may or may not need it.

In Combinatorial mode sharding can be combined with **Continue From File** to resume a previously stopped sharded run, provided the same shard count is used.

Save to Script

The **Save to Script** button writes the dialog's current settings into the script, so that the script itself fully defines the optimization - useful before running with **OptNoDialog** or in **Command Line Mode**, or simply to keep a configured run reproducible. After validating the settings the same way **Run Tests** would, it:

- adds an **OptimizeSettings** section at the end of the script (or updates the existing one) containing every setting that differs from its built-in default, including the Results File when one is set
- removes the section instead, if every setting matches its built-in default
- adds or removes the {X} markers in the **Parameters** section to match the current parameter check boxes, including parameters defined in included files

The changes are made in the script editor window and left unsaved; a single Undo reverts them. A script with more than one OptimizeSettings section, or whose section is in an included file, is not modified - a message explains why.

Script-Defined Settings and No-Dialog / Batch Mode

All of the Optimization Dialog settings have script-level equivalents - see **Optimize Settings** for the full list.

When any *OptimizeSettings* items are specified in the script, their values determine the initial selections in the dialog. Any settings changed in the dialog become the settings used to run the optimization even when they differ from the script's optimize settings.

The special **OptNoDialog** setting can be used to run optimizations without first opening the dialog. When optimizations are run with *OptNoDialog*, no remembered dialog settings are applied. Instead the optimization uses only the script-specified settings or their defaults if not specified.

Scripts run in **Command Line Mode** with **-optimize** always run in no-dialog mode even if not specified in the script.

9.3. Walk-Forward Tests

What a Walk-Forward Test Is

A walk-forward test simulates periodic re-optimization. The overall date range is divided into a series of intervals, and for each interval the strategy is run using the best parameter values found in the *previous* interval. Rather than measuring how a single fixed set of parameters would have performed, it measures the effectiveness of the *process* of re-optimizing at regular intervals.

Two Ways to Create One

A walk-forward model can be generated either from the **Optimizer dialog** or entirely in the script with an **OptimizeSettings** section. Either way, an Optimize run appends a **WalkForward** section to the script, which RealTest then replays whenever the script is run in Test mode. The settings that turn an ordinary optimization into a walk-forward are the same in both places:

(A WalkForward section can also be written by hand: in its simplest two-date form it defines an explicit in-sample / out-of-sample split test with no optimization involved -- see **In-Sample / Out-Of-Sample**.)

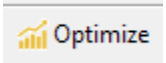
In the dialog	In OptimizeSettings	This example

Optimization Mode	**OptimizeMode**	Combinatorial
Score	**OptScoreCol**	NetProfit
Time Unit	**OptTimeUnit**	Days
Test Length	**OptTestLength**	252
Test Interval	**OptTestInterval**	63
Keep Best	**OptKeepBest**	1
By Date	**OptKeepByDate**	True
Create Walk-Forward	**OptWalkForward**	True

The other dialog options simply control the optimization itself; the items above are what make it a walk-forward.

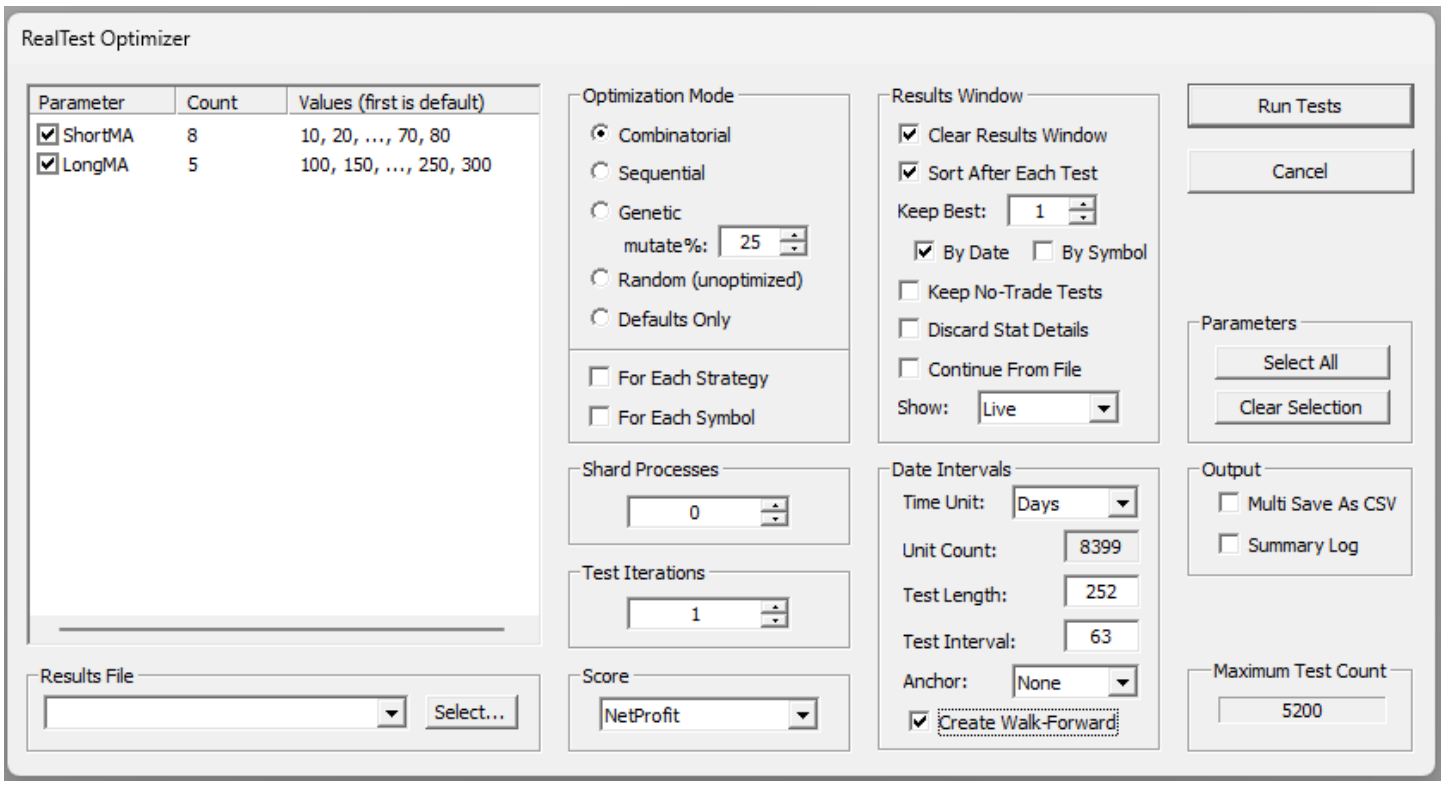
Setting It Up in the Optimizer Dialog

This walkthrough uses the sample2.rts script from **Tutorial 2**.

Open that script again and click on 

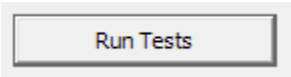
.

Now make all of the following selections in the Optimizer dialog:



1. **Parameters** - Check both parameters.
2. **Optimization Mode** - Set to "Combinatorial"
3. **Test 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 Window** - 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. **Time Unit** - Set 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. **Create Walk-Forward** - Check this.
13. **Maximum Test Count** - This will show how many tests need to be run. (The current version of the example script has different parameters so the count will be different.)

Now you are ready to press 

and let RealTest generate your walk-forward model.

Though there are about 5,000 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 the overall date range

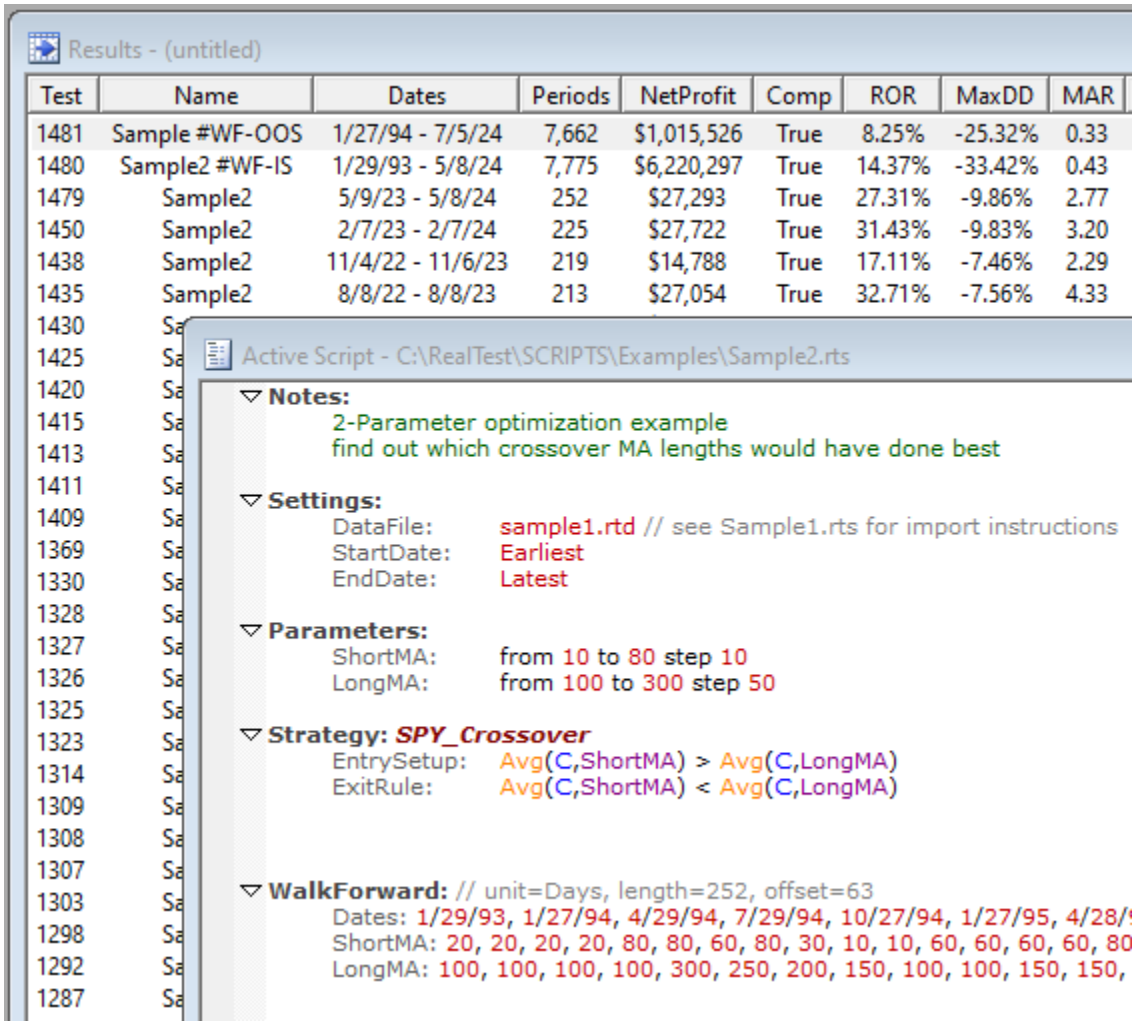
for each of the 40 possible combinations of the two parameters

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:



The Results window shows the best "in-sample" test for each interval. The final two tests run are the in-sample (1480) and out-of-sample (1481) results for the entire date range.

The script window shows the new *WalkForward* section that was generated. These three rows extend far to the right. The first row contains the start date of each interval and the subsequent rows contain the best value found for each parameter in each interval.

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 in *Test* mode.

Setting It Up in the Script

Rather than filling in the Optimizer dialog, you can define the entire walk-forward in the script by adding an **OptimizeSettings** section. The block below reproduces the dialog selections shown above for sample2.rts, with one addition:

```
OptimizeSettings:
   OptimizeMode: Combinatorial
   OptScoreCol: NetProfit
   OptTimeUnit: Days
   OptTestLength: 252
   OptTestInterval: 63
   OptKeepBest: 1
   OptKeepByDate: True
   OptWalkForward: True
   OptNoDialog: True
```

With that section present, run the script in *Optimize* mode (Run / Run Optimization). Because the block sets *OptNoDialog: True*, the Optimizer dialog is skipped and the run starts immediately; the same *WalkForward* section is generated and appended, after which running in *Test* mode replays it exactly as before.

Leave *OptNoDialog* out (or set it to *False*) if you would rather have the Optimizer dialog open pre-filled with these settings, so you can review or adjust them and confirm before launching the run.

The factory example **spy_tlt_uis.rts** is a complete walk-forward strategy set up this way. Open it from the Examples folder to see an *OptimizeSettings* section and the *WalkForward* section it generates in a working script.

9.4. In-Sample / Out-Of-Sample

A common way to guard against curve fitting is to develop and tune a strategy using one date range (the in-sample period) while reserving a later range (the out-of-sample period) to confirm the results on data that development never touched. RealTest can run both periods with a single Test command, adding a separate, clearly labeled row to the Results window for each.

Setting It Up

Add a **WalkForward** section to the script containing a single *Dates* line with two dates: the start of the in-sample period and the start of the out-of-sample period.

```
WalkForward:
   Dates: 1/1/2000, 1/1/2018
```

Running the script in Test mode then runs two tests instead of one:

- *TestName #WF-IS* -- the in-sample test, run from the first date through the second date
- *TestName #WF-OOS* -- the out-of-sample test, run from the second date through the **EndDate** setting

Both ranges are limited to the **StartDate** and **EndDate** settings, and a WalkForward section is ignored when a script is run in Orders mode.

With Parameters

No **Parameters** section is required. If the script does define parameters, the WalkForward section can

list one value for each; both tests then use those values, which also appear in the parameter columns of the Results window.

```
Parameters:
   MALen: from 20 to 50 step 10

WalkForward:
   Dates: 1/1/2000, 1/1/2018
   MALen: 30
```

Selecting the Test Period Per Run

During development you may want to run only the in-sample period, later only the out-of-sample period, or the whole range as a single test -- without disturbing the locked split dates. **Conditional comments** make the choice a one-line edit at the top of the script:

```
// pick one (or none = full range as one test):
//#define IS
//#define OOS
#define SPLIT

Settings:
   DataFile: is_oos_example.rtd
#ifdef IS
   TestName: "RSI Pullback IS"
   StartDate: 2000-01-01
   EndDate: 2017-12-31
#else
#ifdef OOS
   TestName: "RSI Pullback OOS"
   StartDate: 2018-01-01
   EndDate: Latest
#else
   TestName: "RSI Pullback"
   StartDate: 2000-01-01
   EndDate: Latest
#endif
#endif

   // ... Data and Strategy sections as usual ...

#ifdef SPLIT
WalkForward:
   Dates: 2000-01-01, 2018-01-01
#endif
```

In **command line mode**, the scope can be selected without editing the script at all:

```
RealTest -define OOS -test myscript.rts
```

Comparing Every Combination in Both Periods

To compare in-sample and out-of-sample results for every parameter combination -- a model-level robustness check, not a parameter selection step -- run the same optimization once per period into the same results file. With *OptimizeMode: Combinatorial* and *OptKeepBest: 0* (keep every test), each combination gets its own results row with its parameter values in the results columns. Reusing the scheme above:

```
RealTest -define IS -optimize myscript.rts
RealTest -define OOS -optimize myscript.rts
```

Add a *ResultsFile* setting so both runs accumulate in one file, and let the conditional blocks tag *TestName* with the period so the rows are labeled. Rows then pair up by the parameter columns (combinatorial order is deterministic, so row N of each sweep is the same combination). Export with *SaveTestListAs* to compare the periods in a spreadsheet or script. See also **Overfitting Analysis**, which does this comparison in a single optimize run: its Surface view pairs every combination's in-sample and out-of-sample Sharpe at a chronological split and reports their correlation, and its Selection view evaluates the in-sample winner across thousands of symmetric partitions.

A WalkForward section with more than two dates defines a full walk-forward playback, in which parameter values change from interval to interval. Sections like that are normally generated by a

walk-forward optimization rather than written by hand; see **Walk-Forward Tests**.

9.5. Overfitting Analysis

Every optimization report answers the question "which parameter values did best?" It cannot answer the more important question: is that ranking real, or did the optimizer just memorize noise? A parameter grid always has a best cell, even on a random walk. If the ranking is noise, the "best" settings carry no information about the future, and trading them means trading a random draw with extra confidence.

Overfitting Analysis measures this directly. After an optimization completes, it replays the question "which parameter combination would I have picked, and how did that pick actually do?" across hundreds to thousands of in-sample / out-of-sample partitions of the same test period, and reports how often the in-sample winner held up. The method is Combinatorially Symmetric Cross-Validation (CSCV) and the Probability of Backtest Overfitting (PBO), from Bailey, Borwein, López de Prado and Zhu (see References below).

Unlike a **walk-forward test**, which produces one out-of-sample path and keeps only the winner of each window, Overfitting Analysis evaluates every parameter combination on both sides of every split, so it judges the whole family of combinations rather than one survivor, and it produces a distribution of outcomes rather than a single verdict. It also requires no extra data and no additional test runs: the analysis is computed from the daily results of the optimization you already ran, adding at most a few seconds.

The analysis window has two views, chosen with the tabs at its top (or by pressing F again while it is showing): **Selection (CSCV / PBO)**, described first below, asks whether picking the in-sample winner works; **Surface (Walk-Forward Correlation)** asks a complementary question of the same data -- whether the whole parameter surface, good cells and bad, carries from in-sample to out-of-sample (see **Surface View: Walk-Forward Correlation** below).

How It Works

1. During the optimization, RealTest records each test's daily marked-to-market return (the day-over-day change of **S.Equity** for the combined account).
2. The timeline is cut into S equal blocks (the Blocks setting, default 16). A few trailing days may be dropped to make the blocks equal.
3. Every way of choosing S/2 blocks as in-sample (IS) is enumerated; the other S/2 blocks are out-of-sample (OOS). With 16 blocks that is 12,870 distinct splits.
4. For each split, the parameter combination with the best in-sample Sharpe ratio is selected -- the combination the optimizer would have picked if it had only seen that IS data -- and its out-of-sample Sharpe is ranked against all the other combinations on the same OOS data.
5. The per-split outcomes are aggregated into the headline numbers described below.

Sharpe here is the mean divided by the standard deviation of daily returns (annualized for display). It is used for both selection and ranking regardless of the optimizer's Score column -- see Limitations below.

Running It

Interactively: check **Overfitting Analysis** in the Output group of the **Optimize dialog** and set **Blocks** if you want something other than 16, then run the optimization as usual. When it completes, a graph window opens showing the analysis.

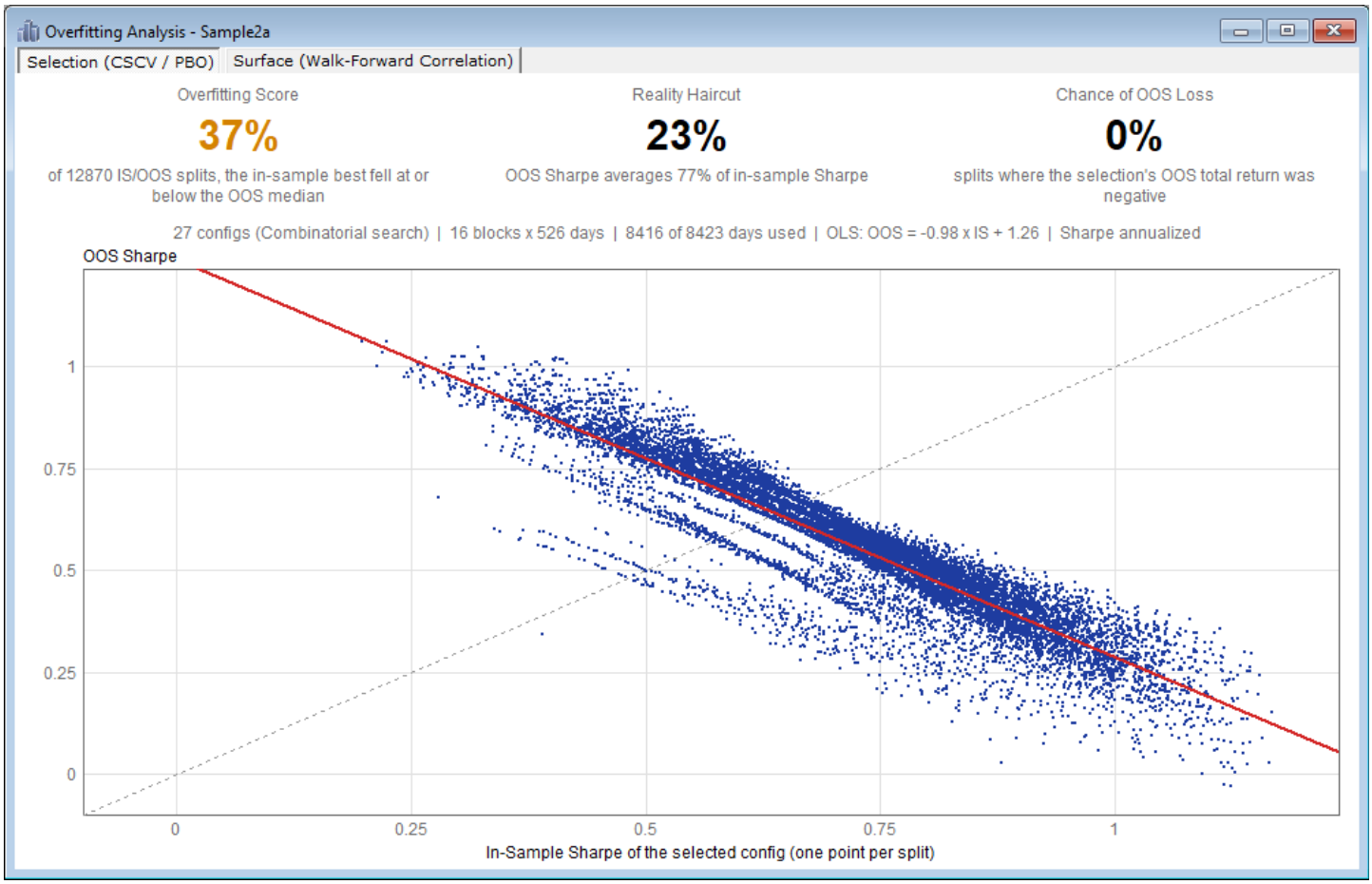
In a script (which also controls batch runs):

```
OptimizeSettings:
   OptOverfit: True
   OptOverfitBlocks: 16   // optional; even, 4 to 20
```

In **command line** batch mode (*realtest -optimize script.rts*), the analysis also writes *overfit.csv* to the OutputPath folder: a summary block (score, haircut, loss chance, fit line, dimensions, warnings) followed by one row per split with its IS and OOS Sharpe, then the Walk-Forward Correlation table (one row per split point) and every combination's IS and OOS Sharpe at the midpoint split. Remember that batch optimizations only run parameters marked *{x}*.

The analysis works with Combinatorial, Random, Genetic and Sequential searches (with a caution for the last two -- see Limitations). It cannot be combined with Walk-Forward, Date Intervals, For Each Strategy / Symbol, Shard Processes, or Continue From File, because those runs do not produce one full-period test per parameter combination; the Optimize dialog will say so.

Reading the Results



Overfitting Score -- the headline: the percentage of splits in which the in-sample best combination landed at or below the median of all combinations out-of-sample (this is the PBO of the literature). Read it as: how often did picking the in-sample winner work out no better than picking at random?

- 50% means coin flip: in-sample rank told you nothing about out-of-sample rank. Optimizing this strategy family is curve-fitting.
- Low values (roughly 25% and below) mean in-sample winners tended to stay winners -- evidence of real, optimizable structure.
- High values (above 50%) mean in-sample winners systematically disappointed -- the classic overfitting signature.

The score is colored green / amber / red at 25% and 45% boundaries. Treat these as orientation, not gospel: the splits reuse the same data and are strongly correlated, so a single optimization's score has limited resolution -- read 43% vs 48% as "about coin flip" rather than as meaningfully different. More combinations sharpen the ranking; very small grids (under 10 combinations) produce a warning because ranks are then coarse.

Reality Haircut -- across all splits, how much of the in-sample Sharpe evaporated out-of-sample: a haircut of 30% means the OOS Sharpe of the selected combinations averaged 70% of their IS Sharpe. Even genuinely robust strategy families show some haircut (selection always captures some luck); values near 100%, or negative OOS altogether, mean in-sample performance was mostly illusion.

Chance of OOS Loss -- the percentage of splits in which the selected combination's out-of-sample total return was negative. A blunt, easily-explained floor: how often would the picked settings have actually lost money?

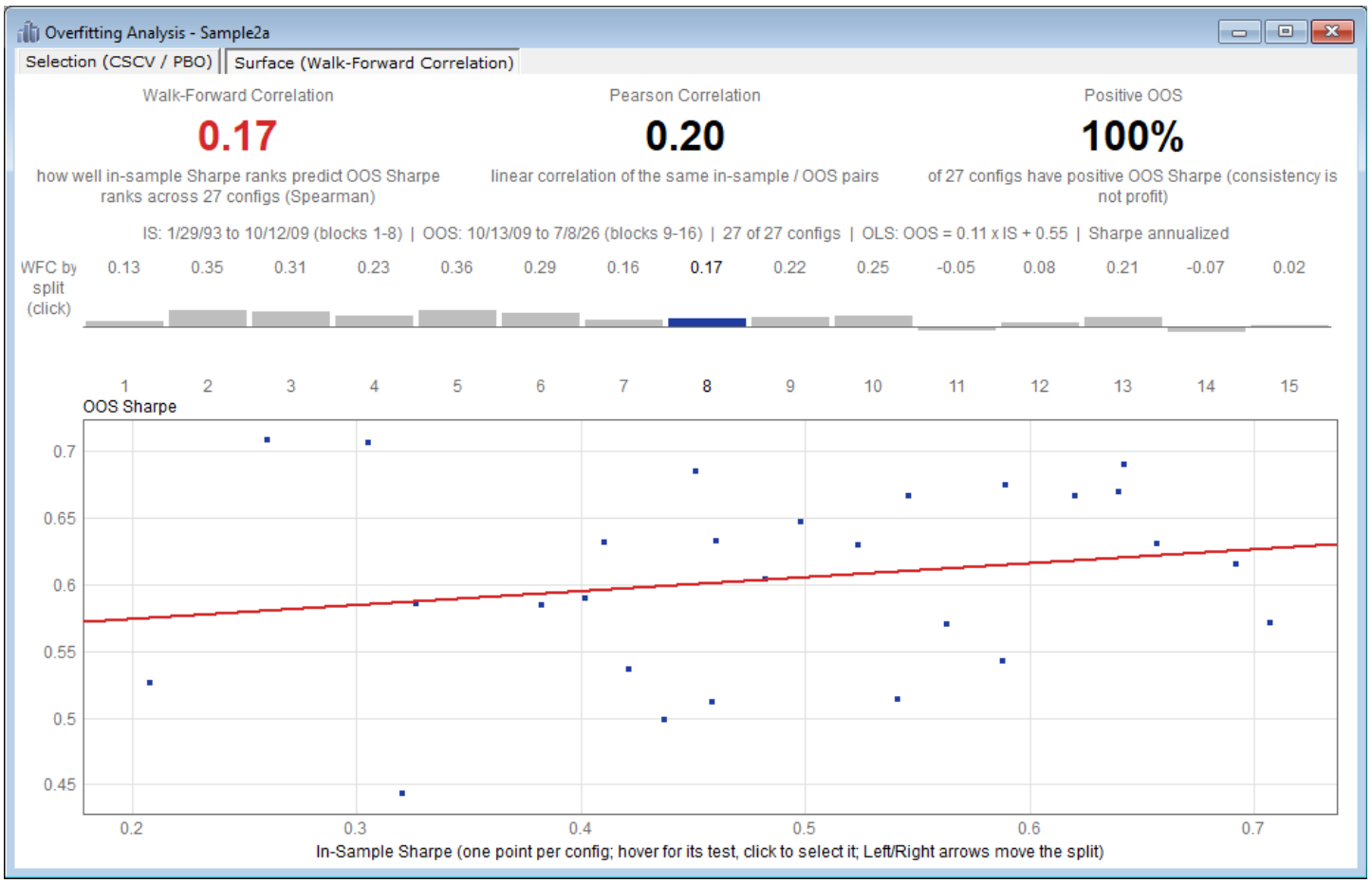
The scatter -- one point per split: x = in-sample Sharpe of the selected combination, y = its out-of-sample Sharpe, with the y = x diagonal and a fit line for reference. The useful reads are where the cloud sits (is OOS performance broadly positive and near the diagonal, or collapsed toward zero?)

and how tightly it hugs the diagonal.

Do not expect a positive fit-line slope, and do not read a negative slope as proof of overfitting. Because the x-axis is always the maximum IS Sharpe, splits where the winner won by more luck sit further right and give more of that luck back out-of-sample -- plain regression to the mean. The slope is therefore negative even for genuinely robust strategy families (and even for pure noise). It is shown for completeness; the Overfitting Score and the Reality Haircut are the discriminating numbers.

As a calibration example: a 60-combination moving-average-cross grid on 33 years of SPY, run with 12 blocks (924 splits), produced Overfitting Score 17%, Reality Haircut 17% and Chance of OOS Loss 1%. Reading: the family has genuine common structure (long exposure to an uptrending index), in-sample ranking carried real information about out-of-sample ranking, and almost no split lost money out-of-sample. The scatter's fit line was nonetheless steeply negative -- the regression-to-the-mean effect described above, not a defect.

Surface View: Walk-Forward Correlation



The Overfitting Score judges one decision -- picking the in-sample winner. The Surface view judges the whole parameter surface instead, using the Walk-Forward Correlation (WFC) diagnostic of Martyn Tinsley (see References). The same blocks are split once, chronologically: the first k blocks are in-sample and the rest out-of-sample (the midpoint split by default). Every parameter combination's in-sample Sharpe is paired with its own out-of-sample Sharpe, and the pairs are correlated across all combinations. If combinations that did well in-sample also did well out-of-sample, and poor ones stayed poor, the optimization surface has persistent structure; if the pairs are a shapeless cloud, the surface is noise and any choice made on it is arbitrary.

Walk-Forward Correlation -- the headline: the Spearman rank correlation of in-sample vs out-of-sample Sharpe across all combinations. +1 means the in-sample ranking was reproduced exactly out-of-sample, 0 means it told you nothing, negative means the ranking inverted. It is colored green / amber / red at 0.5 and 0.2 -- orientation only, since the value depends on the grid you tested (see below). **Pearson Correlation** is the ordinary linear correlation of the same pairs; a large gap between the two usually means a few extreme combinations are steering the linear fit. **Positive OOS** is the percentage of combinations whose out-of-sample Sharpe was positive -- reported alongside because consistency is not profit: a surface whose losers stay losers correlates perfectly and makes no money.

Read the two together: high correlation with mostly-positive OOS results suggests structural edge; high correlation with negative OOS results is a consistent loser; low correlation means the surface did not persist, and any good OOS result among the combinations is as likely to be chance. And read them

against the Overfitting Score: a robust family with a reliable selection rule shows high WFC and a low score; high WFC with a high score means the surface is real but its exact peak wanders (prefer a plateau or an average of neighbors over the single best cell); low WFC with a low score means selection happened to work despite an unstable surface -- investigate before trusting it.

The split profile -- the strip of bars shows the Walk-Forward Correlation at every possible split point (in-sample = blocks 1 to k), so a single split's number can be seen in context: a surface that persists shows positive bars across most of the strip, while a lone high bar is a coincidence of where the boundary fell. The highlighted bar is the split being shown; click another bar, or press the Left and Right arrow keys, to move the boundary. The header line gives the in-sample and out-of-sample date ranges of the current split and how many combinations it includes (combinations that never traded on one side have no Sharpe there and are left out).

The scatter -- one point per combination: x = its in-sample Sharpe, y = its out-of-sample Sharpe, each axis scaled to its own data, with the fit line. Hovering a point shows its test number, its parameter values and the two Sharpes; clicking it selects that test in the results list, so an outlying combination can be examined directly.

Two cautions. First, the value depends on the combinations you feed it: a grid padded with obviously bad settings (very short lookbacks, absurd thresholds) will correlate well simply because the bad cells are bad in every period -- which is why the grid should contain only settings you would consider trading, and why Positive OOS is shown alongside. Second, one split is one sample; a history with a regime change near the boundary can show a low correlation for a family that is otherwise stable, which is what the split profile is there to reveal. The screenshot above (Sample2a's 27-combination moving-average grid on 33 years of SPY, 16 blocks) shows a Walk-Forward Correlation of 0.17 at the midpoint with 100% positive OOS and a profile of mostly small positive bars: a family with a common long-equity edge but only weak persistent ranking among its members -- consistent with its Overfitting Score of 37% on the Selection tab, and a reminder that the two views measure different things.

The Display

The analysis appears as a graph type in the **Optimization Results Graphs** window and behaves like the other types. It opens automatically when the optimization completes, and closing it loses nothing: the most recent analysis is kept until another optimization with Overfitting Analysis replaces it, and can be re-displayed in any optimization graph window with the F key, the ResGraph menu ("Overfitting Analysis"), or the Options dialog.

- **Log Summary Text** (ResGraph menu) writes the headline numbers, followed by a table of the Walk-Forward Correlation at every split point, to an untitled Log window -- handy for keeping a text record alongside other notes.
- **Show Data** (ResGraph menu) opens the current view's points in a list window, from which they can be copied or saved like any other list: the full split table (Split, IS Sharpe, OOS Sharpe) for the Selection view, or every combination's IS and OOS Sharpe at the split being shown (Config, Test, IS Sharpe, OOS Sharpe) for the Surface view.
- **Copy Image to Clipboard** (Ctrl+C), **Save Image as PNG File**, and the Dark Background option work as in the other graph types.

Choosing the Block Count

Blocks (S) must be even, between 4 and 20. The number of splits is the number of ways to choose S/2 blocks from S: 8 blocks gives 70 splits (coarse; fine for a quick look), 12 gives 924 (good for shorter histories), 16 gives 12,870 (the default), and 20 gives 184,756 (finest; the analysis takes a few seconds).

The trade-off: more blocks give more splits (smoother statistics) but shorter blocks, and each block should still contain enough days to make a Sharpe ratio meaningful -- the analysis warns below 20 days per block. As a rule of thumb, keep the default 16 for ten or more years of daily data and drop to 12 or 8 for shorter histories. The block count is shown with the results because the score depends somewhat on it; when comparing strategies, compare at the same block count.

Limitations

- It reuses the same history. Overfitting Analysis detects selection overfitting -- whether ranking combinations in-sample predicted their relative merit out-of-sample within the data you tested on. It is not a substitute for genuinely untouched data, and it cannot detect the overfitting that comes from trying many different strategy ideas until one fits the era you keep testing on.

- Selection is by Sharpe, whatever your Score column. The analysis asks "does picking the in-sample Sharpe winner work?" even if your optimizer scored by NetProfit, MAR, or a custom formula. Path-dependent scores (anything involving drawdown) cannot be computed on the non-contiguous day sets the method requires. In practice a family with no Sharpe-stable ranking will not have a MAR-stable ranking either.
- Genetic and Sequential searches deliberately spend their tests near in-sample winners, so the tested combinations are not a neutral sample and the score can shift. The analysis still runs but flags a warning; Combinatorial and Random runs give the cleanest readings.
- Combinations that never trade have undefined Sharpe; they are never selected and rank below every trading combination. If many combinations in the grid never trade, a warning appears and the score should be read skeptically.
- Returns are the combined account's daily mark-to-market, the same series as S.Equity -- including CashInOut/FeesInOut flows if the script uses them, and frozen at zero after a combination wipes out its equity (also warned).
- A stopped or failed optimization produces no analysis, and runs whose returns matrix would exceed about 1 GB (roughly, tests x trading days x 4 bytes) skip it with a note in the error log.

Example

```
Settings:
   DataFile: sample1.rtd
   BarSize: Daily

OptimizeSettings:
   OptimizeMode: Combinatorial
   OptOverfit: True
   OptOverfitBlocks: 12

Parameters:
   MALen: from 10 to 100 step 10   {x}
   ExitLen: from 5 to 30 step 5   {x}

Strategy: MACross
   EntrySetup: C > Avg(C,MALen)
   ExitRule: C < Avg(C,ExitLen)
```

Run the optimization from the dialog, or headless with *realtest -optimize <script>* (which also writes *overfit.csv*).

References

- Bailey, Borwein, López de Prado, Zhu: *The Probability of Backtest Overfitting* (Journal of Computational Finance, 2017; SSRN 2014).
- López de Prado: *Advances in Financial Machine Learning* (Wiley, 2018), chapters 11-12.
- Tinsley: *Walk Forward Correlation: A Diagnostic for Over-Fitting and Structural Edge in Trading Strategy Optimisation* (SSRN 6324079).

RealTest's implementation follows the papers' CSCV/PBO construction, with selection and ranking by Sharpe ratio of daily returns. The Walk-Forward Correlation view applies Tinsley's diagnostic to the same daily-return matrix, with the same Sharpe metric and with the in-sample / out-of-sample boundary placed at the block boundaries; the split profile is the multi-window extension the paper lists as future work.

10. Analyzing Test Results

After a test runs, RealTest offers several ways to examine what happened. These topics describe them.

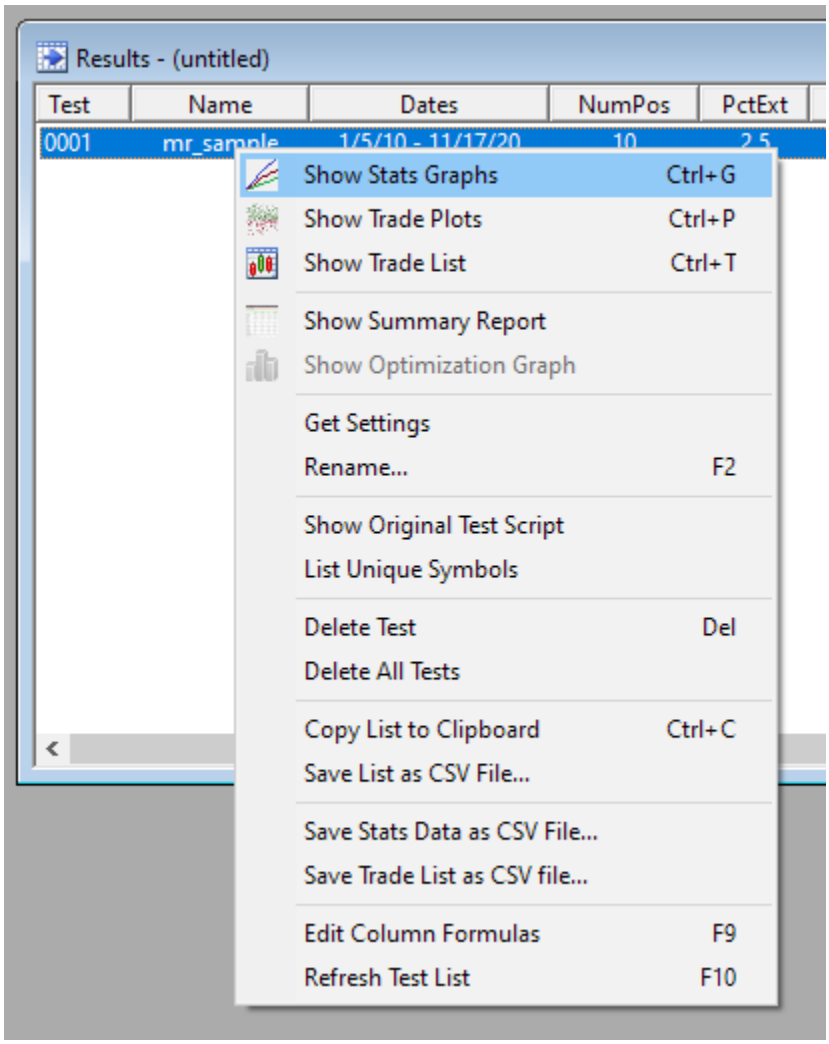
- **The Results Window** - the results row and the detail behind it
- **Test Summary Report** - the formatted summary of a test
- **Test Details Log** - the step-by-step log of a test

10.1. The Results Window

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)

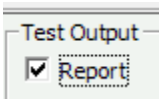
List Unique Symbols creates a new log window with a sorted list of each symbol that was traded at least once in the test.

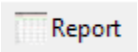
The remaining items control the user interface of the Results window.

10.2. Test Summary Report

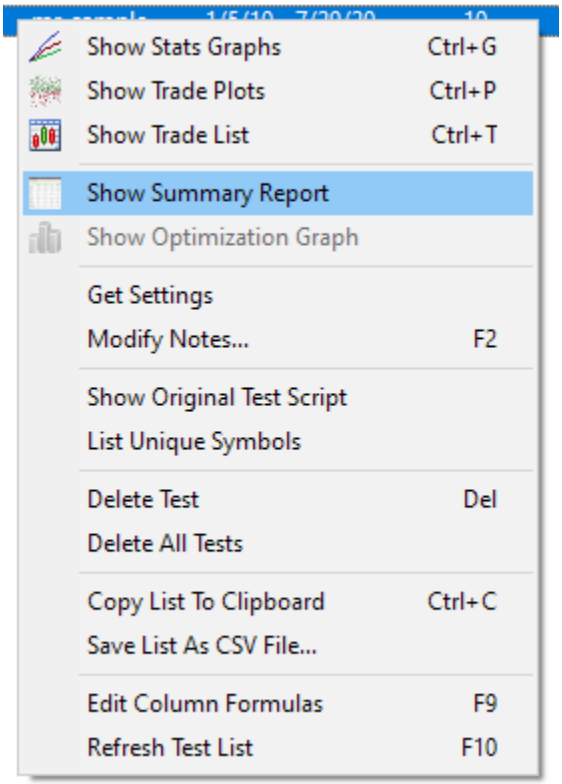
The Summary Report provides an overview of the results of a test in a unified output format.

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 Reports tab of the **Program 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.

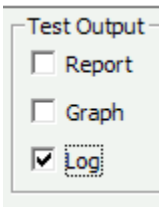
See **Analyzing Test Results** for pointers to some more detailed and powerful ways to study your backtests.

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 {|} in 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.

10.3. 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 

specified, either via the **Settings Panel** or the **Settings** script section. There is not a way to create a test details 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 **Output Path**, 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:

Log - C:\RealTest\Output\Logs\TEST_0003.TXT

DATE	TIME	STRATEGY	INFO	SYMBOL	ACTION	QUANTITY	PRICE	FX	SLIP	COMM
1/3/00	open	sector_etfs	$0 of $100,000 (0.00%) invested in 0 positions							
1/3/00	open	benchmark	$0 of $100,000 (0.00%) invested							
1/3/00	open	sector_etfs	MaxPositions = 5 MaxSameSym = 1							
1/3/00	open	benchmark	MaxSameSym = 1							
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	benchmark	1 Setup: 1=SPY(0);							
1/3/00	open	sector_etfs	5 Orders: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0);							
1/3/00	open	sector_etfs		XLB	buy	752	26.70	1	0.0266	3.76
1/3/00	open	sector_etfs		XLF	buy	841	23.74	1	0.0238	4.21
1/3/00	open	sector_etfs		XLK	buy	371	55.68	1	0.0539	1.86
1/3/00	open	sector_etfs		XLV	buy	647	31.03	1	0.0309	3.24
1/3/00	open	sector_etfs		XLY	buy	643	31.03	1	0.0311	3.22
1/3/00	open	benchmark	1 Order: 1=SPY(0);							
1/3/00	intraday	sector_etfs	$100,732 of $100,000 (100.73%) invested in 5 positions: XLB(752); XLF(841); XLK(371); XLV(647); XLY(643)							
1/3/00	intraday	benchmark	$100,810 of $100,000 (100.81%) invested							
1/3/00	close	sector_etfs	$100,732 of $100,000 (100.73%) invested in 5 positions: XLB(752); XLF(841); XLK(371); XLV(647); XLY(643)							
1/3/00	close	benchmark	$100,810 of $100,000 (100.81%) invested							

Use the **LogDetail** script setting to control which of the available test details will be included in your log files.

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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	DATE	TIME	STRATEGY	INFO	SYMBOL	ACTION	QUANTITY	PRICE	FX	SLIP	COMM	DIVIDEND	PROFIT	ALLOC	COST	M2M	EXPOSURE
2																	
3	1/3/2000	open	sector_etfs	$0 of $100,000 (0.00%) invested in 0 positions													
4	1/3/2000	open	benchmark	$0 of $100,000 (0.00%) invested													
5																	
6	1/3/2000	open	sector_etfs	MaxPositions = 5 MaxSameSym = 1													
7	1/3/2000	open	benchmark	MaxSameSym = 1													
8																	
9	1/3/2000	open	sector_etfs	5 Setups: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0);													
10	1/3/2000	open	benchmark	1 Setup: 1=SPY(0);													
11																	
12	1/3/2000	open	sector_etfs	5 Orders: 1=XLB(0);2=XLF(0);3=XLK(0);4=XLV(0);5=XLY(0);													
13	1/3/2000	open	sector_etfs		XLB	buy	752	26.7	1	0.0266	3.76			$100,000	$20,077	$0	20.08%
14	1/3/2000	open	sector_etfs		XLF	buy	841	23.74	1	0.0238	4.21			$100,000	$40,045	$0	40.04%
15	1/3/2000	open	sector_etfs		XLK	buy	371	55.68	1	0.0539	1.86			$100,000	$60,702	$0	60.70%
16	1/3/2000	open	sector_etfs		XLV	buy	647	31.03	1	0.0309	3.24			$100,000	$80,779	$0	80.78%
17	1/3/2000	open	sector_etfs		XLY	buy	643	31.03	1	0.0311	3.22			$100,000	$100,732	$0	100.73%
18	1/3/2000	open	benchmark	1 Order: 1=SPY(0);													

11. Trading Your System

Once your strategies are ready, RealTest can help turn them into live orders. These topics describe the tools involved.

- **Before You Go Live** - paper-trade first, and what RealTest can and cannot do
- **Tomorrow's Orders** - generating the orders for the next session
- **Generated Order Types** - the kinds of orders RealTest produces
- **OrderClerk** - the OrderClerk order-management helper
- **CSV Order Baskets** - exporting orders as broker basket files
- **IB Rebalance Tool** - rebalancing an Interactive Brokers account

11.1. Before You Go Live

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 can be used as part of a full trading automation solution but, by itself, cannot be used to place orders directly to any brokerage.

The other topics in this section show the various ways that RealTest can help with your live trading.

11.2. 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:

Log - C:\RealTest\Output\Orders\mr_sample_20201118_orders.txt

```
*** 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 setting*ExitTime* to *ThisClose* for these strategies):

```
Log - C:\RealTest\Output\Orders\mr_sample_20201118_orders.txt
*** 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**.

11.3. 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 are 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.

DataSource to IB Futures Symbol Mapping

RealTest includes a file called **ibfutures_ndu.csv**. This file maps **Norgate** futures symbols to the corresponding IB symbols.

Similarly, the **ibfutures_csi.csv** file provides mappings for **CSI** futures symbols to the corresponding IB symbols.

Note that RealTest does not generate CSV futures orders with contract month letters embedded in the symbol. Rather, it includes the contract's expiry date separately in the "Expiry" column.

To use a futures symbol mapping file, you must add e.g. **SymChangeList**: **ibfutures_ndu.csv** to your script's **Settings** section.

11.4. 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.

Since RealTest 2.0.32.3 and OrderClerk 1.0.29.5, each order also carries the symbol's asset ID (when the data source provides one, e.g. Norgate), which OrderClerk stores with the trade and returns in the trade list. This lets RealTest keep tracking a position whose symbol has changed since entry; see **Symbol Changes in Open Positions**.

For more information about OrderClerk and how to use it with RealTest, see the separate *OrderClerk User Guide* documentation which can be found in the installation folder after you've installed OrderClerk.

11.5. Symbol Changes in Open Positions

When you trade a system live, a position can be entered under one symbol and need to be exited under another. Between entry and exit the data vendor may rename the stock (e.g. FB became META), delist it, or trade it temporarily under a different exchange code. Your trade list still carries the symbol as it was at entry, so RealTest has to reconcile the two when it **plays back the trade list** to establish your current positions before generating **Tomorrow's Orders**.

This topic explains what RealTest does automatically, what the **OrderClerk** round trip adds, and the two cases that still need a line in your script.

What happens if a symbol is not found

If a trade list symbol cannot be matched to any symbol in the loaded data file, the trade is dropped with the log message *TradeList: symbol XYZ not found in currently loaded data file*. In an order run this means the position is simply not there: no exit order will be generated for it, and the strategy could even enter it again as a new position. The message appears in the Log window and in *errorlog.txt* during order runs, and in the **SaveLogAs** file on every run when that setting is used, which is the easiest way for an automated workflow to detect the situation.

Delisted symbols

When a stock has been delisted since the trade was entered and the data file includes delisted stocks

(e.g. a **Norgate** import with delisted securities), RealTest finds the delisted version of the symbol automatically, e.g. *CATM* is found as *CATM-202105*. Nothing needs to be configured for this case.

Renamed symbols: automatic handling by asset ID

Data sources with persistent asset IDs (Norgate, or any source where you provide an *AssetId* column in a **Symbol Information File**) allow RealTest to follow a symbol change automatically. This requires RealTest 2.0.32.3 or later and OrderClerk 1.0.29.5 or later, and works as follows:

- Every order RealTest generates carries the asset ID of its symbol, in the *AssetId* column of the order list (the *assetid* tag in row 2 of *OrderClerkTemplate.csv*; see **CSV Order Baskets**). The template in the RealTest program folder already includes it; if you keep a custom copy of the template in your OrderClerk folder, add the column there.
- OrderClerk stores the asset ID with the trade when the entry order is loaded (the *AssetId* column of *OrderClerkTrades.csv*) and hands it back to RealTest as part of the trade list.
- On every order run RealTest matches open trades by asset ID first (**TLFields** *AssetId*, which takes precedence over the symbol). If the data vendor's symbol for that asset ID has changed, RealTest rewrites the trade's symbol and logs *TradeList: symbol for AssetId 2088 changed from HGO.au to HGODB.au*. The position is never lost and exit orders continue to be generated.
- When such an exit order is loaded into OrderClerk, it recognises the asset ID under a new symbol and renames the trade (both the RealTest symbol and the broker symbol) so that fills and position reconciliation still match. OrderClerk reports this as *Symbol for AssetId ... changed from ... to ...* in its messages.

Nothing needs to be configured for this; it only requires that both programs were at the versions above when the **entry** order was generated, because that is when the trade acquires its asset ID. Open trades entered before then have no asset ID and still match by symbol. If you want them covered too, type the IDs into the *AssetId* column of *OrderClerkTrades.csv* by hand (with OrderClerk closed). The asset ID of any imported symbol is available in RealTest as **InfoID**, e.g. as a column in a **Data Scan**.

Temporary exchange codes (ASX consolidations, US "D" suffix)

Some symbol changes are temporary, and the broker does not follow them. The most common example is an ASX consolidation (reverse split): for about two weeks after the effective date the shares trade under a deferred-settlement code, and Norgate renames the symbol accordingly, e.g. *HGO.au* becomes *HGODB.au* and then reverts to *HGO.au*. Interactive Brokers keeps trading it as *HGO* throughout. A similar thing occasionally happens with US stocks (mainly OTC) which carry a "D" suffix for a period after a reverse split.

The asset ID mechanism is not the right tool here: RealTest would keep the position and generate the exit order, but the order would go to the broker under the temporary code and be rejected. Instead, use the **IncludeList** alias syntax to import the renamed symbol under its original name for as long as the temporary code lasts:

```
Import:
    IncludeList:    HGODB.au>HGO.au
```

With this line the trade list, the generated order symbol (*HGO*), and the splits file for the consolidation (see **Split Handling**) all continue to work unchanged. If your import also pulls the temporary symbol in by itself (e.g. via a Norgate watchlist), you will have the same stock twice; keep the strategy from entering the second copy by adding *and ?Symbol <> "HGODB.au"* to its EntrySetup. An **ExcludeList** cannot be used for this, because it would remove the aliased import as well. Remove the alias line once the symbol has reverted.

Manual mapping with SymChangeList

For any other case, such as a permanent rename with a data source that has no asset IDs, or old trade lists from before asset IDs were recorded, list the change in a **SymChangeList** file (*symchanges.csv* in the OrderClerk folder or the RealTest program folder is found automatically). This is applied when the trade list is read and is an alternative to editing the trade list itself.

Spotting changes early

Add **SaveLogAs** to your Settings. Trade list warnings (symbol not found, symbol changed by asset ID, delisted symbol replaced, unusable rows) are then written to that file on every run, so an automated

order workflow can check it each morning and alert you before an exit order is missed.

11.6. 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 as well (or 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.csv*, where your actual test name and the current date are used.

You can override this default by adding **OrdersFile** to your settings and/or to specific strategies 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.

Note that with OrderClerk it is not necessary to specify an OrdersTemplate path. RealTest includes a standard OrderClerkTemplate.csv file in its installation folder and will use that file automatically when generating orders for OrderClerk. The default OrderClerkTemplate.csv includes a *split* column so that OrderClerk (1.0.29.3 or later) can correctly handle position sizes after stock splits, and an *AssetId* column so that OrderClerk (1.0.29.5 or later) can pass each trade's asset ID back to RealTest, which allows symbol changes to be handled automatically when using Norgate data or any other source with persistent asset ID values.

The IB Basket template looks like this:

Action	Quantity	Symbol	Exchange	Currency	TimeInFor	GoodTilDa	GoodAfte	OrderTyp	LmtPrice	AuxPrice	OcaGroup	OrderId	ParentOrc	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:

Order List - C:\RealTest\Output\Orders\mr_sample_20220215.csv

#	Action	Quantity	Symbol	Exchange	Currency	TimeInForce	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
rtsym	original symbol as shown in RealTest
assetid	the symbol's asset ID if the data source provides one (e.g. Norgate), else blank
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)
allhours	1 if order is active outside regular hours (sets OutsideRTH), 0 for regular hours only
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 9
nocomma	suppresses the comma in the CSV output of this column
tradeid	round-trip trade ID (required for OrderClerk, not needed otherwise)
uniqueid	a specific artificial order ID that is kept unique across sessions

If you require blank columns, simply add them to the header row but leave them blank in the second row of the template.

11.7. 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.

12. Other Ways to Use RealTest

Beyond running scans and tests interactively in the RealTest window, there are several other ways to put RealTest to work:

- **Using an Imported Trade List** - analyze a trade list generated outside RealTest
- **Using Command Line Mode** - run imports, scans, tests, and orders in batch
- **Using the Automation API** - drive a running RealTest from scripts and AI tools over HTTP
- **Using Multiple Instances** - run more than one copy of RealTest at once
- **Using RealTest with Agentic AI** - an AI-assisted approach to strategy research

12.1. Using an Imported Trade List

Overview

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**.

To play back an imported trade list, create a **Strategy** that includes a **TradeList** statement.

Trade List Types

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.

Trade List Format

For either type, the trade list must be a plain text file in standard US CSV format.

The field delimiters must be commas and the decimal points must be periods.

The CSV file must begin with a header row containing its column names.

If the column names are not the standard names as shown in the example below then you must provide a **TLFields** definition to tell RealTest what field is in each column.

The *DateIn* and *DateOut* columns can use any supported **date format**.

If your **program options** date format setting conflicts with the trade list's date format (i.e. m/d/y vs. d/m/y), use **TLDateFormat** to override the setting while parsing this trade list.

A trade whose date falls on a non-trading day (a weekend, a holiday, or any date not present in the imported data) is moved to the prior trading day. Warnings identify each trade this happened to, and any open position row whose details can't be used at all. The warnings appear in the log window (always for order runs; with **TestOutput***: Log* or **SaveLogAs** for tests), and an order run also notes them in the text order list.

Other Trade List Settings

RealTest expects quantities and prices in a trade list to be actual "as-traded" values, i.e., not split-adjusted. If your trade list has been adjusted, use **TLAdjusted** to let RealTest know this.

If your trade list dates and times are known to be in the wrong time zone (e.g. your local zone rather than the exchange zone), use **TLTimeShift** to correct this.

If you want trades labeled for a particular strategy to play back in a strategy with a different name, use **TLStratName** to specify this.

Example Script

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 Strategy Rules

In addition to simply playing back a list of trades to generate stats and graphs or look at trade 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. (You must use **EntrySkip** rather than **EntrySetup** to override a trade list entry. *EntrySetup* is ignored in overlay mode.)

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 **all** exits from the trade list.

Switching from Trades Playback to Strategy Rules

By default any strategy containing both a *TradeList* statement and some *Strategy* formulas is played back as described above by overlaying any specified strategy elements on the imported trades.

It is also possible to run a "hybrid" playback where the trade list runs verbatim up to a particular date and then the strategy rules take over entirely after that date. To activate this mode, add **TLEndDate** to your Strategy definition.

Using Hybrid TradeList+Formula Strategies for Tomorrow's Orders

The behavior described in the two prior sections applies when the **run mode** is *Test.*

When a strategy with a trade list is run in **Orders Mode**, it is assumed that the purpose of the trade list is to establish current positions and account value from actual live trades playback prior to generating tomorrow's orders.

Therefore playback is always done in *hybrid mode* when generating orders.

All strategy element definitions (including *TLEndDate* if specified) are ignored throughout the entire playback (as when *TLEndDate: Latest* is specified in *Test* mode). The strategy definition is only applied at the final order generation step.

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 (if needed) *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.

12.2. 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 -optimize *script.rts***

- runs *script.rts* in **Optimize** mode
- use special {X} comments in **Parameters** to specify which ones should be optimized
- use **OptimizeSettings** in your script to specify other optimization options

❖ **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

❖ **realtest -parse *script.rts***

- checks the *script.rts* for syntax errors and does nothing else

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
9	a requested output file (SaveTradesAs, OrdersFile, etc.) could not be written

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. The error message is also written to the *stderr* command line output stream.

In addition, a file called **batchlog.txt** is created in the RealTest folder for each batch-mode run. Unlike errorlog.txt (which keeps growing), batchlog.txt is recreated each time, making it easier for AI agents and other automated tools to process. If the script parsed and ran without errors, batchlog.txt will contain an "OK" success message. A run that completes but fails to write an output file the script asked for (a Save...As CSV, or the OrdersFile of an orders run) is not a success: it exits with code 9 and writes no "OK" line.

Syntax errors reported in batch mode use script line and column numbers rather than raw byte offsets, which is more useful when working with AI agents such as **Claude Code**.

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 Error code was %errorlevel%

: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.

The command **-define xxx** can be used to define a symbol on the command line. This enables **#ifdef** / **#endif** blocks in a script to be controlled from outside the script. Multiple -define arguments can be specified. This is especially useful when doing AI-driven research, e.g. with **Claude Code**. Note that -define by itself does not engage command line (batch) mode, so it can also be used when opening a normal interactive session; batch mode engages only when a run-mode argument such as -test is present.

The commands **-api** and **-port N** enable the **Automation API** for this launch (the second also chooses its port). Neither implies batch mode: RealTest opens normally and can then be driven over HTTP by scripts and AI tools, which is the better choice when the same instance will run several tests or when a task needs the interactive windows.

The command **-hide** starts RealTest minimized without activating its window, so launching it never takes keyboard focus away from the application you are working in. Like -api it does not imply batch mode; it is intended for automation programs that launch their own RealTest instance, and the two are typically used together.

12.3. Using the Automation API

RealTest can expose a local HTTP + JSON interface that lets scripts and AI tools drive a running RealTest window: run scripts, read results, send menu commands, capture screenshots, and more.

This is an alternative to **Command Line Mode**. Both run the same scripts and write the same output files; they differ in how RealTest is controlled.

Batch Mode or the API?

- **Use batch mode** for a single unattended run: a scheduled morning order generation, a nightly import, a shard of a large optimization, or a one-off test. Each run is a fresh process with an exit code and a batchlog.txt, and there is nothing to connect to or shut down afterwards.

- **Use the API** when the same instance will be asked to do several things: iterating on a script (loaded data and calculated Data columns stay in memory, so repeated tests skip the load and calculation phases and typically run several times faster), reading results without parsing CSV files, or getting parse errors back as structured line and column numbers rather than log text.
- **Use the API** for anything only the interactive window can do: Trade Plots and Factor Scan, Charts, Graphs, any menu command, and screenshots of those windows.

The AI research materials in the Research folder (see **Using RealTest with Agentic AI**) describe both approaches to the assistant and let it choose.

Enabling the API

The listener is off by default. Turn it on for every session in realtest.ini, or for one launch on the command line:

Setting	Default	Meaning
Api	0	in the [Options] section of realtest.ini; 1 starts the listener whenever RealTest starts with that ini file
ApiPort	7827	the TCP port to listen on (127.0.0.1 only); 0 picks any free port
ApiToken	(empty)	an optional secret; when set, every request must include it in an X-RealTest-Token header
-api		command line argument: enable the API for this launch regardless of the ini setting
-port N		command line argument: enable the API on port N (0 = any free port) for this launch

Neither argument puts RealTest into batch mode; the window opens normally and the API simply becomes available once startup is complete. If the port is already in use, RealTest logs the failure to errorlog.txt and runs without the API.

An automation script that launches its own RealTest should also pass **-hide** (start minimized without taking keyboard focus) and **-inifile** with a copy of realtest.ini, so that the driven instance saves its settings to the copy rather than to your own configuration. In that copy, set AskSaveUntitled, AskUnloadData, AskDeleteTests and ConfirmStop to 0 so no Yes/No prompt can appear. Make a fresh copy before each launch: on exit the instance saves its session back into the copy, including the list of open scripts, so a reused copy reopens them.

```
realtest.exe -api -hide -inifile C:\temp\drive.ini C:\Scripts\myscript.rts
```

Finding a Running Instance

Each listening instance writes a small JSON file named after its process id to *%LOCALAPPDATA%\RealTest\api* with its port, URL, executable, ini file and start time, and deletes the file when it exits. A program that launched RealTest reads the file for its process id; a program attaching to an already running session lists the folder. Until the instance has finished starting, connections are refused, so retry until a GET /status request succeeds and reports state "idle".

How Requests Work

Requests are ordinary HTTP GET and POST calls to http://127.0.0.1:port, with JSON bodies and JSON responses. Every JSON response is either *{"ok":true,"result":...}* or *{"ok":false,"error":{"code":...,"message":...}}*. Tabular data (results, trades, stats) is returned as CSV text.

Requests that change the program state are carried out on the RealTest window thread in the order received, and each response is sent only when RealTest is idle again: no run in progress and no calculation still working. Callers therefore never need to wait or retry between steps. A request that arrives while a run is in progress is refused with a "busy" error rather than queued. Read-only requests such as status and log are answered immediately, even during a run.

A run request creates a **job**. By default the response waits until the job has finished; a "wait" value in seconds caps the wait, after which the response reports the job as still running and it can be polled. The job object reports success, the batch-mode exit code, every error message RealTest would have shown, structured parse errors (file, line, column, message), the tests it added (count and test-number range), the Results window that received them, and any modal dialog it is waiting on.

Requests

Request	Purpose
GET /status	run state, busy flag, active script, data file, status bar text, current job
GET /help	a machine-readable list of all requests
POST /open	open a file as File > Open would (script, results, CSV, trade list)
POST /run	run a script in a given mode (parse, test, optimize, import, scan, orders, apply), optionally with #define names for that run; an optimization runs without the Optimize dialog
GET /jobs, GET /jobs/{id}	remembered jobs, or one job (optionally waiting for it to finish)
POST /stop	request Stop for the running job, or dismiss a modal dialog that is holding it
GET /log	the Log window contents as numbered lines, or as plain text
GET /results	the Results window as CSV, the same columns as SaveTestListAs
GET /results/{n}/trades	the trade list of test n as CSV, in the Full layout by default or the Compact layout on request
GET /results/{n}/stats	the daily stats of test n as CSV, as SaveStatsAs writes it
GET /windows	the open child windows: id, class, title, active
POST /activate, POST /close	make a child window active, or close it
POST /select	select a row in a list window (a Results test by number, or any row by index) so that menu commands act on it
POST /command	send any menu command by its command id; returns when RealTest is idle again
POST /key	send a keystroke (arrow keys, Enter, F-keys, a character) or type text into a child window
POST /screenshot	capture a child window or the whole frame as a PNG file or image response
POST /exit	close RealTest gracefully

Menu commands act on the active child window and its selected row, exactly as they do when chosen from the menu, so activate the intended Results, Plot or Chart window (and select the intended row) before sending one. Commands that open file dialogs should be avoided; use /open and the script Save...As settings instead.

Security

The listener accepts connections only from the local machine (127.0.0.1). Programs running under your own Windows account already have the same access to your files as RealTest does, so the remaining concern is a web page in your browser making blind requests to localhost. Such requests are rejected: RealTest refuses any request carrying an Origin header, requires the Host header to name 127.0.0.1 or localhost, and requires POST bodies to be declared as JSON. If other user accounts share the computer, set ApiToken so that requests without the token are refused.

Error Messages and Prompts

While RealTest is carrying out an API request or job, and at all times in an instance launched with -api or -port, error message boxes are not shown. The text is returned in the job's messages and appended to batchlog.txt and errorlog.txt, so a driven instance never blocks on a message box. A session that merely has Api=1 in its ini keeps its normal message boxes for your own actions. Yes/No questions are still shown, which is why a driven instance should run with the Ask... and Confirm... options turned off in its ini copy. If one appears anyway, GET /status reports it (state "dialog", with the question text) and POST /stop dismisses it. The Optimize dialog never appears for a run started through the API: the optimization runs as if the script had OptNoDialog: True. Should RealTest crash, the request that was running is answered with a "crashed" error, GET /status reports state "crashed" with the crash report

and the dump file's path, the log can still be read, and POST /exit closes the crash message box, which ends the process.

Learn More

The complete request reference, with every parameter and response field and working PowerShell, curl and Python examples, is **realtest_api.md** in the Research folder inside your RealTest installation (see **Using RealTest with Agentic AI**). It is written so that an AI assistant can read it and drive RealTest without further help.

12.4. 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 has 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:

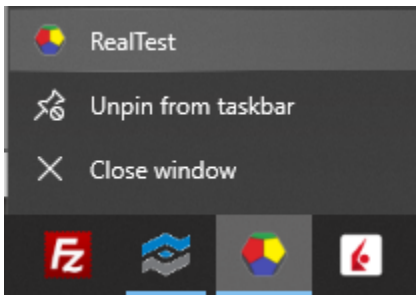
1. The RealTest.ini file, which is used to remember things like window positions, default settings and recent file lists, is automatically saved when RealTest is closed, so whichever instance is closed last will be the one whose settings etc. are saved in RealTest.ini.

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.

12.5. Using RealTest with Agentic AI

An agentic AI coding assistant can write, run, and iteratively improve RealTest scripts through natural-language conversation. You describe a strategy in plain English; the assistant writes the *.rts* script, runs RealTest to parse and test it, reads the results, and refines from there -- all in one conversation.

To make this work well, RealTest ships a **research folder** of ready-made materials that teach an AI assistant how to write and run RealTest scripts. These materials were developed and tested primarily with Anthropic's Claude Code, but they are plain Markdown and work just as well with other capable agentic coding tools.

The RealTest Research Folder

Starting with version 2.0.31.7, the RealTest installer creates a ready-to-use research folder at *Scripts\Research* inside your RealTest installation directory (e.g. *C:\RealTest\Scripts\Research*). It contains:

- **CLAUDE.md** -- instructions that teach an AI assistant how to write and run RealTest scripts: the command-line workflow, common pitfalls, and iteration discipline. Claude Code reads this file automatically when you start a session in the folder; with another tool, point the assistant at it (or copy its contents into whatever instruction file that tool uses).
- **realtest_script_language.md** -- a concise and complete RealTest script language reference, written by studying the RealTest source code. CLAUDE.md tells the assistant to consult it as needed.
- **userguide** -- the complete RealTest user guide in Markdown form (the same content you are reading now), one file per topic. Start at *userguide\toc.md*, which lists every topic with a link to its file. This lets the assistant answer questions about menus, dialogs, settings, and program behavior -- not just script syntax.
- **research_output.rts** -- an Include file that defines a consistent Results column set and adds **SaveTestListAs** and **SaveTradesAs** settings to streamline iterative strategy research.
- **realtest_api.md** -- the reference for the **Automation API**, which lets the assistant drive a running RealTest over HTTP instead of launching a fresh batch process for every run. CLAUDE.md explains when each approach is the better choice.

Choosing an AI Assistant

Any agentic AI coding assistant that can read local files and run programs on your computer can use these materials. The reference setup -- and the one the research folder was built and tested against -- is **Claude Code** from Anthropic.

To use Claude Code, download the desktop app for Windows from **https://claude.ai/download**. It includes everything you need -- no terminal experience required. If you already have the Claude Code CLI installed and prefer working in a terminal, that works too -- the desktop app and CLI are the same product with different interfaces. On first run you will authenticate with your Anthropic account, and Claude Code will read CLAUDE.md automatically when you open a session in the research folder.

Claude Code offers two models: **Opus** and **Sonnet**. Switch between them with the model dropdown in the desktop app, or the /model command in the CLI.

Opus is recommended for RealTest work. It handles the multi-step workflow (write script, parse, read errors, fix, test, interpret results) more reliably and with less back-and-forth, which may actually use fewer tokens despite its higher per-token cost. Sonnet can work for simpler tasks but is more likely to thrash on parse errors or misread results. The "medium" effort level setting is generally sufficient.

Competing models and their agentic coding tools also work well with the research-folder materials. Because the instructions, language reference, and user guide are all plain Markdown, you can point any capable assistant at them. A tool that looks for its own instruction filename rather than *CLAUDE.md* can simply be told to read it, or you can copy its contents into whatever file that tool uses.

Granting Permissions

Agentic tools ask permission before accessing files or running programs, and how you grant standing permission varies by tool. In Claude Code, your first session in the research folder will frequently ask before accessing local files. Sometimes "Always Allow" is an option. If it keeps asking the same permission after you say "Always Allow", ask it to give itself blanket permission for that operation (e.g. running RealTest) on any file. It will then modify its own permission list file, which you can view and even edit yourself in the .claude folder it creates inside the research folder. See the Claude Code documentation on **settings.local.json** for the full rule syntax.

An alternative to the specific-permissions dance is to put Claude in "Auto" mode, which is smarter about bypassing permission queries for low-risk operations while still occasionally asking for something more questionable.

Your First Session

1. Open your AI coding assistant (for Claude Code, the desktop app or a terminal)

2. Point it at your research folder (*C:\RealTest\Scripts\Research*)

3. (optional, Claude Code) Turn on "Auto" mode to avoid the permissions dance

4. Try a prompt like:

Write a mean-reversion strategy for S&P 500 stocks that buys 3-day pullbacks and exits on a bounce. Run it and show me the results.

The assistant will write the .rts file, run RealTest to parse and test it, read the results, and summarize the performance -- all in one conversation.

Tips

- Be specific about your universe and time frame -- "Russell 1000 stocks since 2000" gives better results than "US stocks"
- Iterate in conversation -- "tighten the stop to 5%" or "add a VIX filter" and the assistant will modify and re-run
- Say what you want to explore, not how to code it -- the assistant handles the RealTest syntax
- If the assistant hits a parse error, it reads the error log and fixes it automatically

13. Backtest Engine Details

This section explains the internal mechanics of the RealTest backtest engine. Understanding these details helps in writing strategies that behave as intended and in interpreting test results correctly.

Topics in this section

1. **Backtest Loop Overview** - how the daily loop processes exits and entries across three time-of-day phases, and how setups are selected and allocated across strategies
2. **Asset Allocation and Position Sizing** - how allocation, sizing, and exposure are managed
3. **Capacity Constraints** - MaxPositions, MaxExposure, and other limits
4. **Compounding** - how equity growth is reinvested
5. **Split Handling** - how stock splits affect price and volume data
6. **Dividend Handling** - how dividends are processed during a test
7. **End-of-Data Exits** - what happens to open positions when a symbol's data ends before the test does
8. **Intraday Fills With Daily Bars** - how limit and stop orders are filled within a daily bar
9. **Intraday Fill Sequence Assumptions** - why intraday entries are processed before exits
10. **Bar Sizes and Multiple Timeframes** - weekly, monthly, and other bar sizes
11. **Calendar Alignment** - how bars are aligned across different markets
12. **Calculation of Trade Excursions** - how MAE and MFE are computed
13. **The Current Bar in Formula Evaluation** - what "current bar" means in different contexts
14. **Specifying a Time Stop** - time-based exit patterns
15. **Number of Bars Required for Functions and Indicators** - minimum data requirements
16. **Scaling In or Out of Positions** - pyramiding and partial exits
17. **Referring to Past Trades in Strategy Formulas** - using trade history in entry/exit logic
18. **Testing Multi-Currency Strategies** - currency conversion and FX handling
19. **Using CSI Futures Data** - CSI-specific data considerations

13.1. Backtest Loop Overview

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.

Test Initialization

Before the daily loop begins (unless using **Dynamic Sizing** mode, which re-evaluates sizing each bar):

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

Daily Loop: Three Phases

For each date in the test range, trading is processed in three time-of-day phases. The order in which exits and entries are processed within each phase is designed to model realistic daily trading as closely as possible:

Phase 1 - At the Open

Process **exits** first, then **entries**. Exits are processed first to free capital for new positions being entered at the open.

Phase 2 - Intraday

Process **entries** first, then **exits**. This is the *opposite* of the other phases. Since the sequence of intraday fills is unknown, the engine cannot assume that exiting one position would free capital in time for another entry on the same bar. To avoid this unrealistic assumption, entries are processed before exits.

See **Intraday Fill Sequence Assumptions** for more detail.

Phase 3 - At the Close

Process **exits** first, then **entries**. Same as the open: exits free capital for new entries.

After all three phases, daily stats are updated and allocation is recalculated for each strategy.

Which phase a given entry or exit executes in is determined by **EntryTime** and **ExitTime**, as well as whether the entry or exit uses a limit or stop price that would be filled intraday.

Exit Processing

In each phase, exits are evaluated for all open positions. A position is exited if any of the following are true:

1. **ExitRule** evaluates to TRUE and **ExitTime** matches the current phase
2. **ExitLimit** or **ExitStop** price was touched without ambiguity (exit at that price)
3. ExitLimit and/or ExitStop price were touched with ambiguity and the **Ambiguity** setting permits exit anyway
4. This is the last bar of data or last date of the test (exit at close, subject to **EndOfTestExits**)

Entry Processing

Entry processing uses a multi-pass approach to fairly allocate capital across multiple strategies while respecting all capacity constraints.

Setups are processed in ascending **EntryPhase** order: the three passes below run for each phase in turn (by default, each benchmark gets its own phase ahead of all strategies), and the actual entry fills happen only after the last phase. A section in a later phase therefore sees the final order state of all earlier phases when its own setups are evaluated.

Pass 1 - Setup Detection

For each strategy, loop through all stocks and evaluate **EntrySetup**, making a list of stocks for which it returns TRUE (setups). Sort this list by **SetupScore** and truncate at **MaxSetups** (if specified).

Pass 2 - Constraint-Aware Setup Selection

Setups are selected in a round-robin fashion across strategies to ensure fair allocation:

1. For each **turn** (loop from 1 to the most setups in any strategy):
   a. For each strategy in **StrategyScore** sequence (highest score first):
   1. Select the next unprocessed setup (ranked by SetupScore)

2. Check if adding it would exceed any *Max...* constraint for the strategy, any **StatsGroups** it belongs to, or **Combined**
3. If no constraints are exceeded, add the setup and update all constraint values; otherwise skip it
4. If **MaxPerTurn** setups have been added on this turn, yield to the next strategy

Pass 3 - Order Finalization

For each strategy, now that the complete setup list is known with all constraints accounted for:

1. Sort setups by **EntryScore** if provided
2. Evaluate **QtyFinal** for each setup to optionally adjust the order quantity
3. Evaluate **EntrySkip** if provided and skip the entry if true
4. Enter the position if **EntryLimit** and/or **EntryStop** have been touched or were not specified

Key Points

1. Position sizes for constraint-checking purposes are always based on **OrderPrice**, not on **FillPrice** (which can't be known yet at setup-processing time)
2. All constraints are applied to the setup list prior to processing any entries
3. Setups for strategies that enter with limit or stop orders count towards all constraints whether or not they are filled
4. This model assumes that all orders are placed in advance of the market open each day with no real-time order management (e.g. live cancellation when a capacity is reached)

13.2. 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.

See **DynamicSizing** for information on how*Quantity* can be used differently in this special alternative strategy mode.

Per-Strategy Equity and Drawdown in Combined Tests

In a combined-compounding test (the default), each strategy's *S.Equity* series is its starting equity plus its own cumulative profit and loss, while its position sizes are driven by the equity of the whole account (*Allocation: Combined(S.Equity)*). Two consequences of this combination surprise people when they graph the individual strategies of a combined test:

- **An individual equity line can go negative.** Because a strategy's position sizes grow with the combined account rather than with its own results, its cumulative dollar losses can exceed its own virtual starting balance, taking its individual *S.Equity* line to or below zero. This is not an error. In combined mode a strategy's "own account" is only a bookkeeping series - the combined equity is the account, and it is the line that reflects what the whole system did.
- **Drawdown percentages are relative to the account, not to the strategy's own equity line.** Per-strategy percentage stats such as **S.MaxDDPct** use **S.Alloc** as their denominator, so in a combined test a strategy's reported drawdown measures what its losses cost the total account. A strategy can therefore show a modest MaxDD% even though its own dollar equity line went negative - the two numbers answer different questions.

To see a strategy's performance in its own terms, graph its **S.TWEQ** (time-weighted, "growth of $1") series, which remains meaningful even when the dollar equity line has gone negative. Alternatively, model each strategy compounding in its own separate account by giving each a **StartPercent**, so that sizes are driven by each strategy's own equity.

13.3. 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 invested
- **MaxLongExp** - maximum percent long invested
- **MaxShortExp** - maximum percent short invested
- **MaxNetExp** - maximum net long-short invested
- **MinNetExp** - minimum net long-short invested
- **MaxInvested** - maximum dollars invested
- **MaxLongInv** - maximum long dollars invested
- **MaxShortInv** - maximum short dollars invested
- **MaxNetInv** - maximum net long-short dollars invested
- **MinNetInv** - minimum net long-short dollars invested
- **MinFreeCash** - minimum free cash in account
- **MaxNewPos** - maximum new position count
- **MaxNewExp** - maximum new percent invested
- **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)
- **MaxEntries** - maximum position entries per day (deliberate selection bias)

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.

(The one exception to the above statement is *MaxSameSym.* This defaults to 1 in each strategy but is infinite in combined strategy groups.)

Most of the above formulas may be included in the definition of a **Strategy**, a **StatsGroup**, or the **Combined** system. 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.

To apply 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 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 selection 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.

When strategies are assigned to different **EntryPhase** values, the selection process runs once per phase, and capacity usage accrues across phases within the bar - setups selected in an earlier phase count against all shared constraints when later phases make their selections.

One analogy for the capacity constraint process 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 iteration.

13.4. 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., ((current_equity - max_equity) / 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., ((current_equity - max_equity) / start_equity)

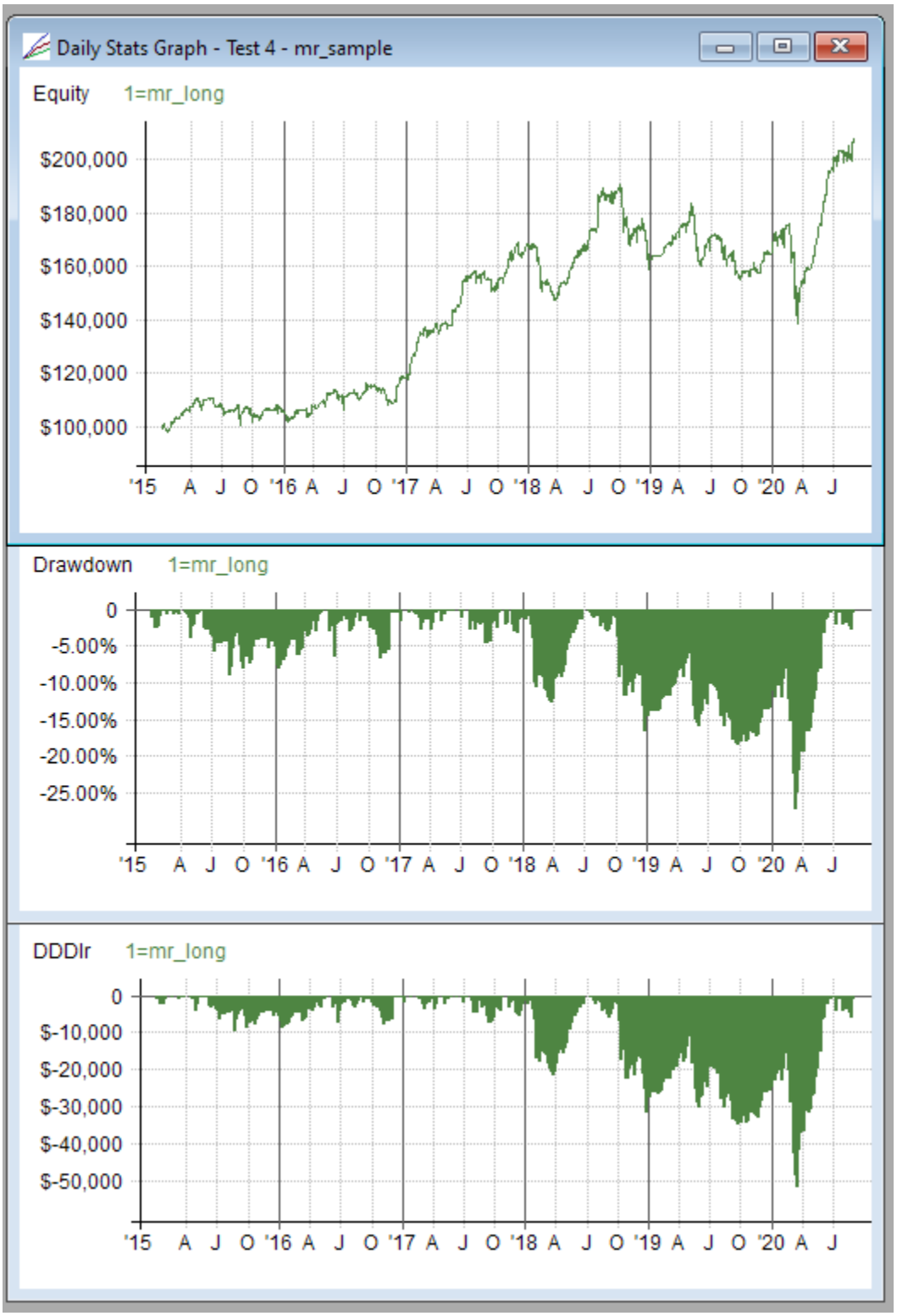
This automatic distinction makes it convenient to use either compounded or non-compounded equity without the typical stats distortions which are described at length in the following paragraphs.

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).

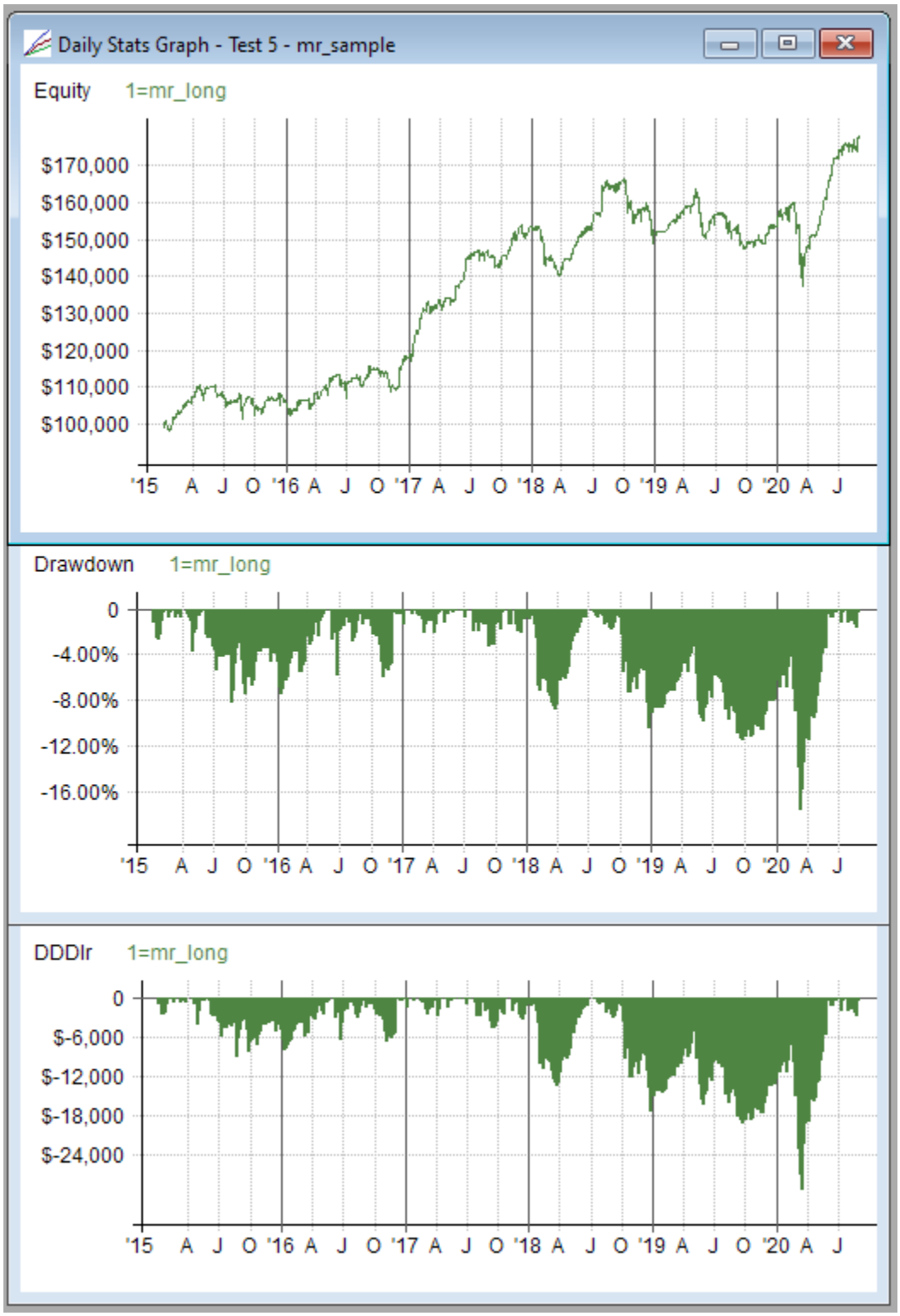
To illustrate, consider the following example.

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).

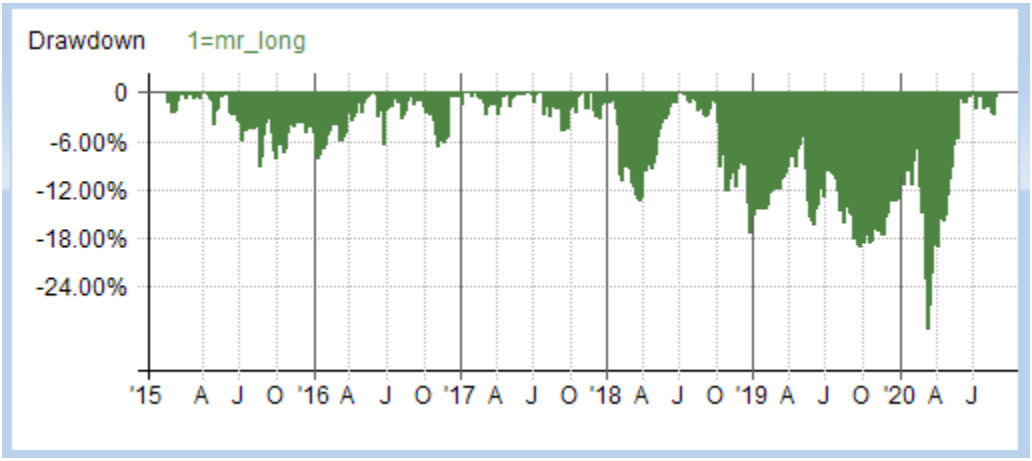
Now if we run this test using non-compounded position sizing (always invest $10K in each position), the results look like this (after temporarily changing graphs.rts to make it ignore the *compounded* flag and show compounded-style drawdown for this result):



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.

13.5. Split Handling

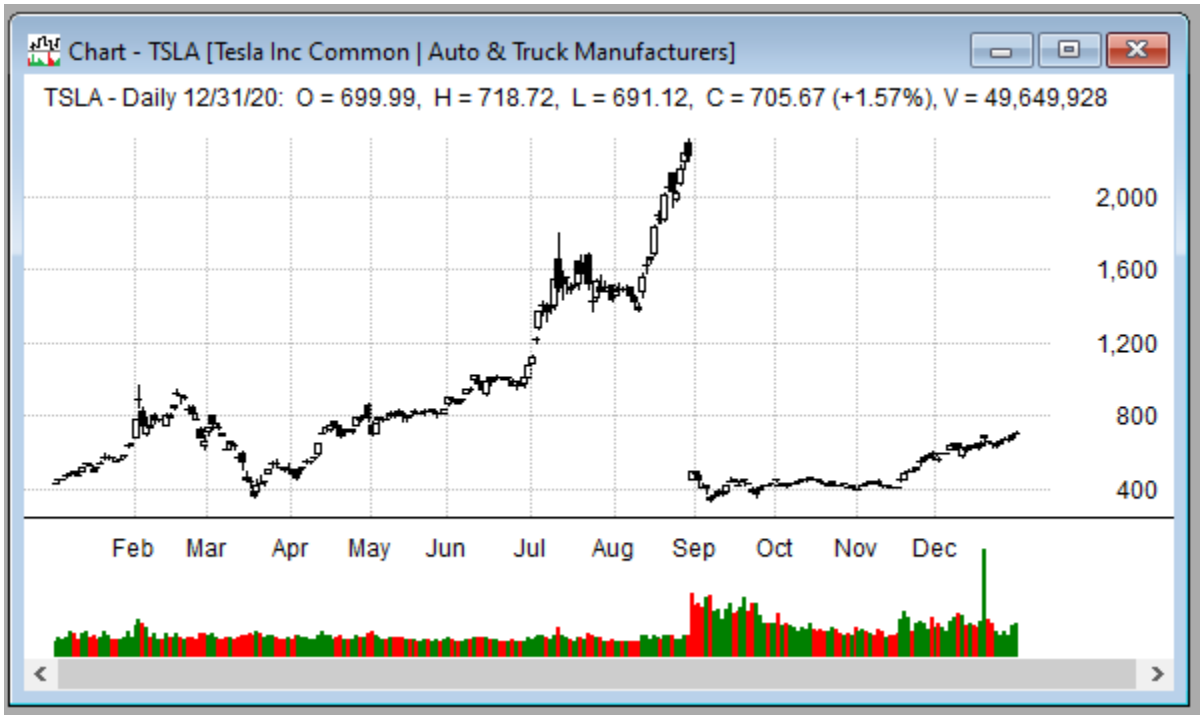
RealTest internally keeps price and volume data split-unadjusted. Any time any formula refers to "Close", the value returned is the real as-traded close for the current bar being evaluated.

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.

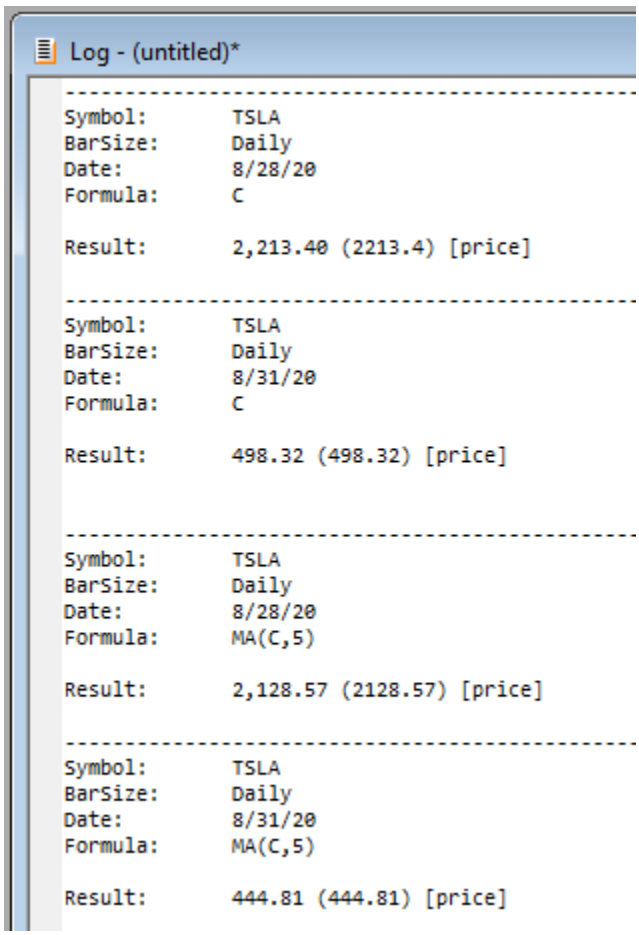
As a simple example, consider TSLA stock in 2020.

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.

In certain rare cases (including the *exponential slope* calculation example above), even when *Log(C)* is kept in the original formula, if it is calculated in **one pass** in the *Data Section*, the values may become distorted around splits. This is because the internal circular-buffer that enables one-pass calculation of things like *Slope* is equivalent to pre-calculating *Log(C)* as a separately-stored item. If you encounter this type of distortion (often easily visible on a chart if you plot the indicator), prefix your data item with **#SlowCalc** to force it to not use the one-pass calculation optimization.

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.

13.6. 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
948	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
.06	-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
3	131.46%	$127,560.95	152.77%	-5.43%	99.95%	$97,033	$0.0000	$0.0000	$11,694.05
4	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
2	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
6	17.11%	$60,924.42	20.65%	-0.15%	100.00%	$356,168	$0.0000	$0.0000	$11,111.14
7	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
2	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
3	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
9	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.

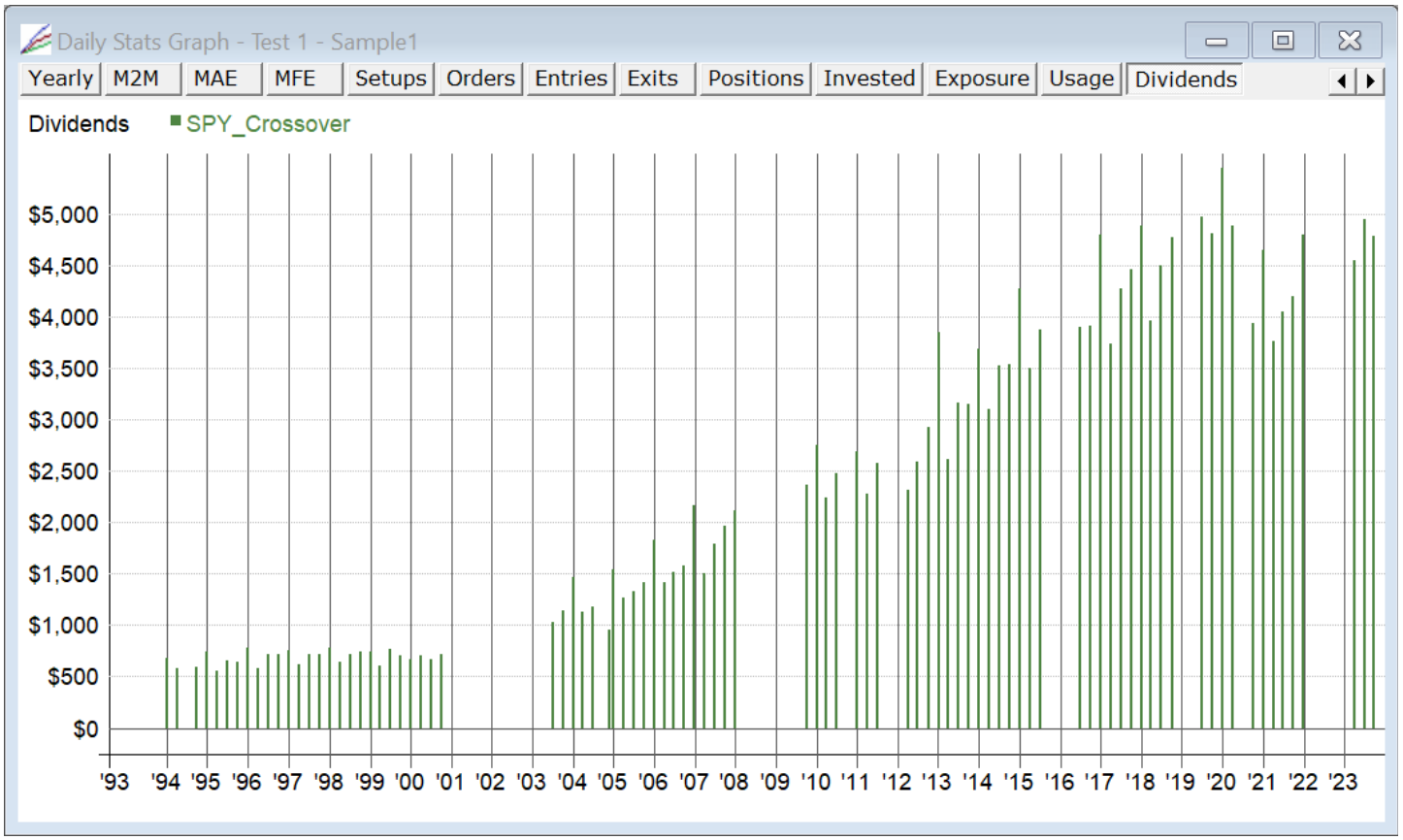
To display the total of dividend amounts received, uncomment this line in the default Results.rts script:

```
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.

13.7. End-of-Data Exits

When a symbol's data ends before the test does, RealTest normally force-closes any open position in that symbol at the symbol's last bar, with exit reason "end of data". This prevents a position from riding a frozen last price through the remainder of the test. This topic explains exactly when that exit happens, when a position is instead held to the end of the test, and how to interpret what you see in the trade list.

When the Exit Happens

The decision is driven by the symbol's delisting date (available in formulas as **InfoDelist**):

- **Strategy positions** get the end-of-data exit if the symbol has no delisting date at all, or has one that has been reached. If the symbol has a delisting date later than its last bar - typically a stock that is halted but still listed - the position is instead held at its last price to the end of the test and closed by the normal "end of test" exit. (Before version 2.0.31.6, every such position got the end-of-data exit.)
- **Benchmark positions** get the end-of-data exit only when the symbol's delisting date has been reached (since version 2.0.32.3). A **Benchmark** with no delisting date is held to the end of the test rather than flattened. Benchmarks exist to be referenced - by **Extern**, **Tracker** and comparison statistics - so a trailing calendar gap (e.g. a US holiday on the latest bar of a mixed-country import) should not flatten a SPY benchmark and disturb the strategies that

reference it.

- **Trade list playback positions** (**imported trades** being played back, including hybrid order runs) are exempt from end-of-data exits unless the symbol's delisting date has been reached.

Where the Delisting Date Comes From

- **Norgate** reports a "last quoted date" for every symbol, which RealTest imports as the delisting date. A genuinely delisted stock has it equal to its final trading day, so an open position in it gets a normal end-of-data exit. A halted-but-still-listed stock has a current last quoted date but no recent bars - that is the held-to-end-of-test case described above, which is intentional: the stock isn't dead, it just isn't printing bars.
- **CSV and other imports** take the delisting date from the DelistDate column of a **SymInfoFile**. Without one, the symbol has no delisting date, so open strategy positions always get end-of-data exits.
- **Yahoo** provides no delisting information, so Yahoo-sourced positions always get end-of-data exits.
- **Futures** contracts use their expiry date in the same role (see **InfoExpiry**).

What You See in the Trade List

A forced exit at the symbol's last bar shows **Reason** "end of data". A position held to the end of the test shows reason "end of test" - but note that its exit date is still the symbol's last bar date, not the test end date, since that is the last date for which a price exists. A trade that appears to exit "end of test" weeks before the end of the test is therefore not an anomaly; it is an open position in a symbol whose data ended (e.g. a halted stock) while its delisting date had not been reached.

Interactions

- **Import padding** - with **Padding** enabled, a short trailing gap in a symbol's data is filled with padded bars up to the latest imported date (unless the symbol's delisting date has passed), so end-of-data handling never comes into play for such gaps.
- **CalendarSym** - when a strategy has a **CalendarSym**, the "end of the test" used in these checks is the calendar symbol's last date.
- **Order runs** - when **generating orders**, the text order list includes a notice when a position (including a benchmark position) recently got an end-of-data exit, so that a symbol whose data has gone stale is noticed before live orders are placed.
- **End-of-test exits** - the **EndOfTestExits** setting controls whether positions still open at the end of the test are closed at all; it does not affect end-of-data exits.

13.8. 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. A pure doji bar (Close = Open) is resolved as in Stop mode.

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.

13.9. 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.

13.10. 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 supported.

When daily bar data is imported, weekly, monthly, quarterly and yearly bars are automatically built. Any of these bar sizes is available to use in your scripts and/or display on charts.

There is a global bar size setting that is specified either on the **Settings Panel** or in the **Settings** script section. 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 use of multiple bar sizes, see **combined_multi_bar_size.rts** in the **Examples** folder.

13.11. Calendar Alignment

RealTest is able to combine data from multiple sources and/or for multiple asset types.

During data import, a "global calendar" is created. This is simply a list of all dates for which any symbol has a bar.

When tests or scans are run, the outermost loop is across this global calendar. For each date, RealTest then loops through the symbols and processes each symbol that has a bar for that date. This ensures that all references to bar data are correctly aligned to the date being processed.

The following is an overview of how calendar logic is applied during backtesting:

```
for each date
   for each symbol
      if it has a bar for this date (the potential transaction date)
         get prior date from the global calendar
         if it had a bar for that date (the order generation date)
            determine and process trading signals
```

What this means, by default, is that for a transaction to be simulated in a backtest, the symbol must have bars for both the current and the prior date in the global calendar.

A strategy can optionally specify **AllowMissingBar: True** to remove the second requirement above. That would allow orders for tomorrow to be generated for symbols currently in trading halts, and backtests to model such orders having been generated. This implies that **capacity** slots might be consumed by stocks that cannot be traded tomorrow, which is why the default is *False* for this setting.

When one script includes data from multiple countries that have different holidays, there are two other solutions available:

1. Add a **CalendarSym** specification to each strategy definition
2. Add a **Padding** specification to your **Import** definition

Option 1 is generally preferable as it avoids potential distortion of multi-bar indicator calculation caused by padding bars, but there may be cases where you need option 2.

Independently of the above, when **Padding** is in effect, RealTest also extends each symbol's most recent bars forward to the global calendar's latest date (bounded by the PadTrailMaxDays ini setting), so a symbol that merely missed the final holiday bar stays aligned for order generation and subsequent backtests.

13.12. 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 at 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

13.13. The Current Bar in Formula Evaluation

When writing a formula that refers to bar data elements such as "High" or "Close", it is important to 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 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:

EntryTime	EntrySetup Current Bar
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:

ExitTime	ExitRule Current Bar
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:

ExitLimitTime or ExitStopTime	Formula Current Bar
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.

13.14. 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.

13.15. 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).

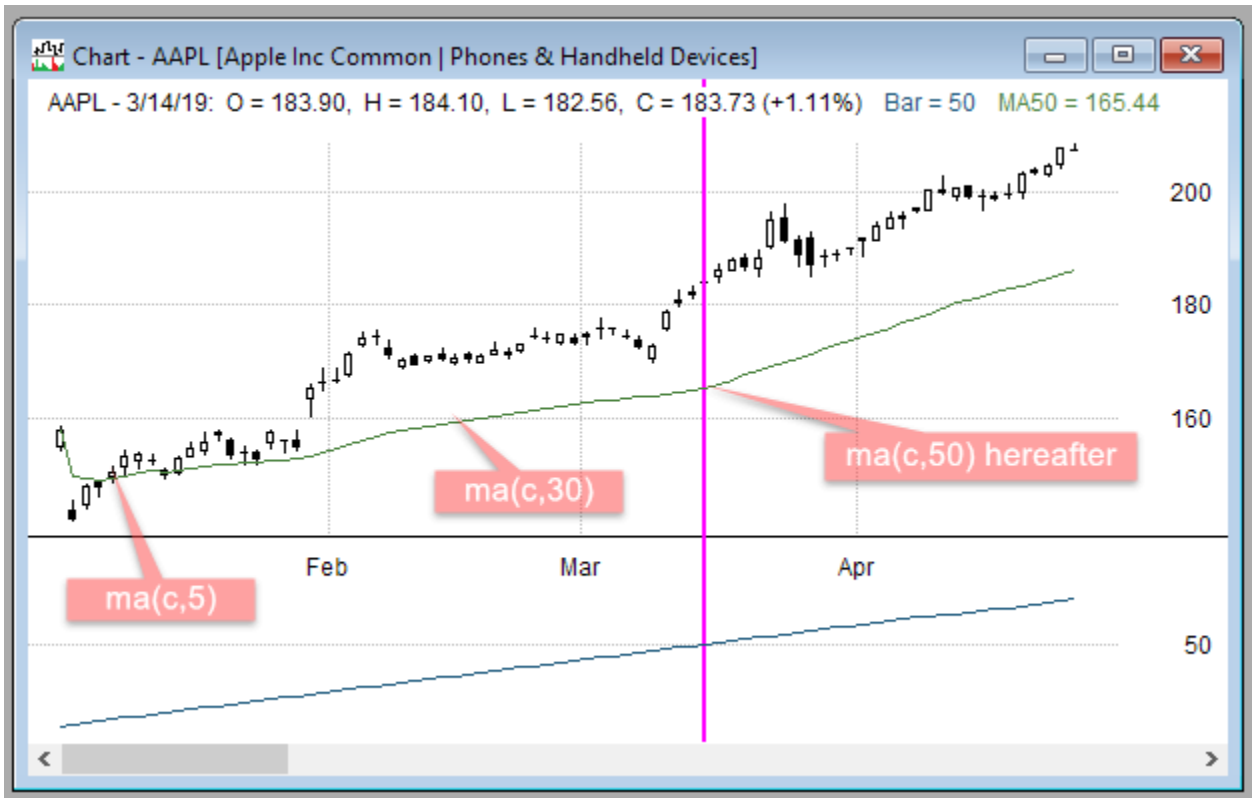
As mentioned above, there is also an option to automatically shorten moving average and indicator lengths when there are not enough bars available. Most of the time, the default setting (off), as described above, is recommended.

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).*

13.16. 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 (thus "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 i**s 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.

13.17. 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.

13.18. 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:

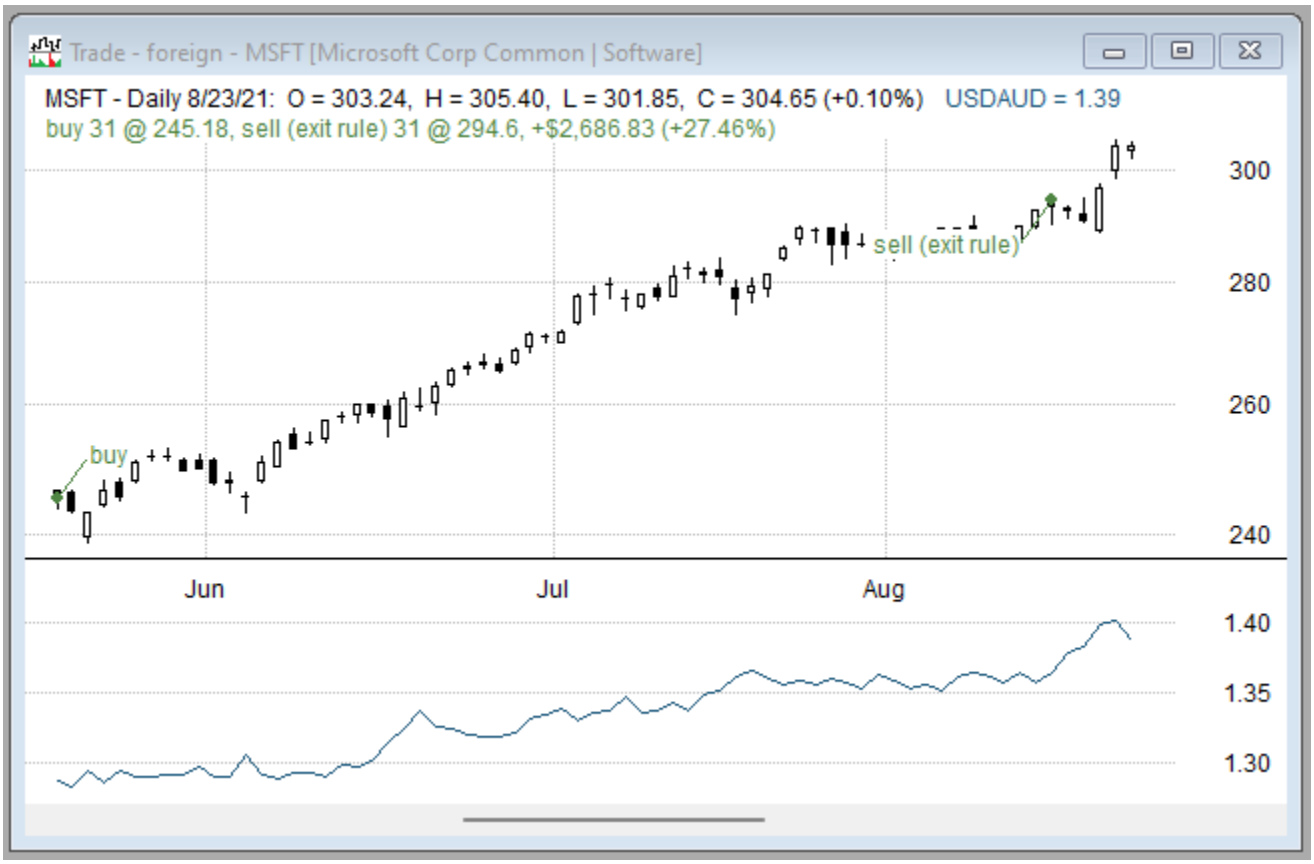
Trade List - Test 1 - 1 Trade

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 shares) * (0.56 dividend per share) * (1.2837 USDAUD ratio on the record date) = $22.28 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.
- When neither currency is USD, the direct pair is not required if both currencies' USD pairs are imported. For example, to trade AUD stocks in a EUR account, you can import AUDUSD and EURUSD instead of EURAUD, and RealTest computes the cross rate from the two USD pairs. The direct pair is used when it is available.
- 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.
- The model described above is for stocks. If you trade a currency pair directly (for example EURAUD in a USD account), its metadata must have a *Type* of *Forex* (with *Currency* set to the quote currency, AUD in this example). Forex and futures profit is the points gained times the

quantity, converted to the account currency at the exit exchange rate, and *T.NetFx* is always zero. Without the Type, RealTest treats the pair as a stock denominated in its Currency and charges the change in exchange rate against the full position value, so a trade that gained in points can show a loss. A *TickSize* (e.g. 0.00001) should also be provided for forex symbols from sources other than Norgate, since the default tick size of 0.01 rounds fills to two decimal places.

- 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)

13.19. Using CSI Futures Data

When using CSI futures data, RealTest is able to backtest strategies and generate trading orders from back-adjusted continuous contract series.

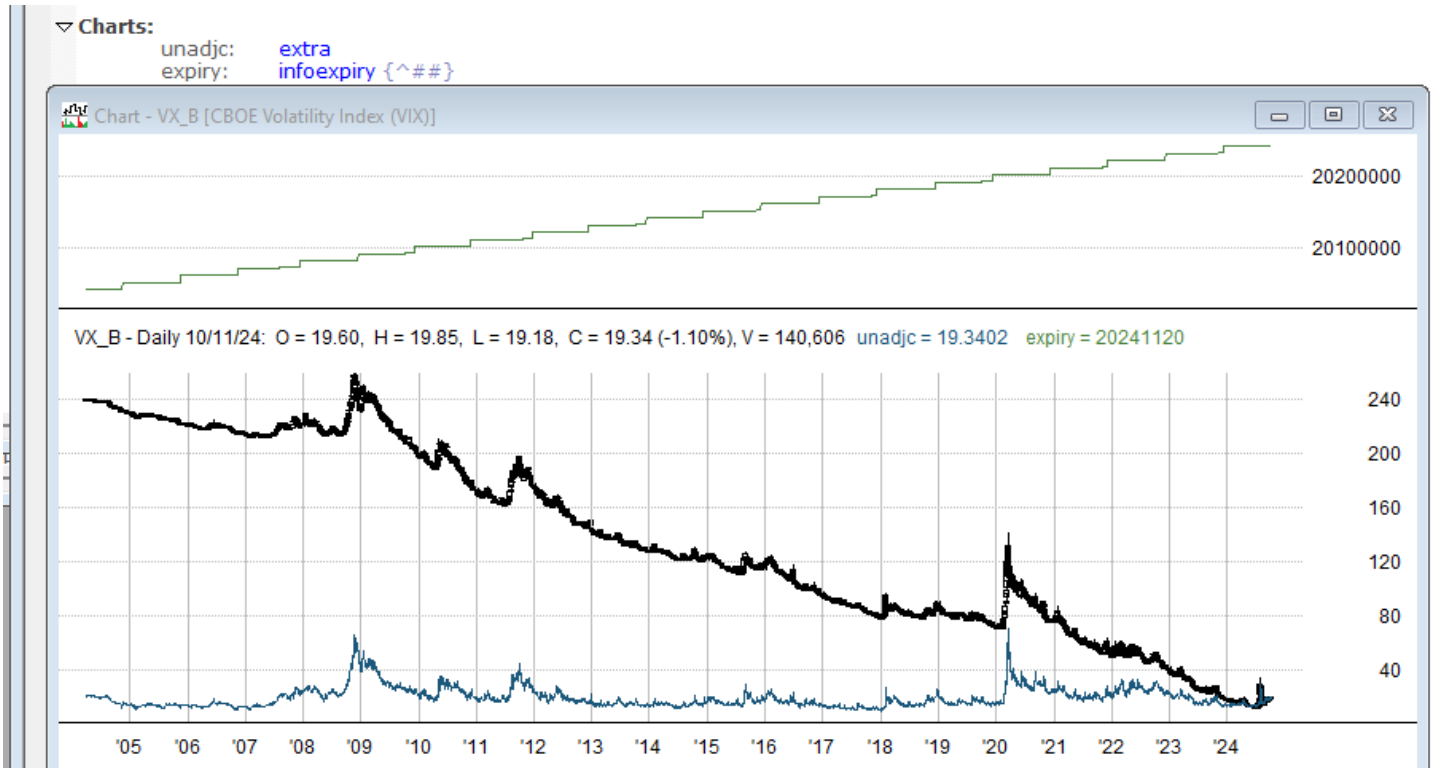
As with *Norgate* futures data, RealTest also fully supports backtesting on the individual contracts with CSI data, as described **here** and shown in example scripts such as f*utures_volume_rank.rts*.

For most use cases, it will prove to be simpler and just as effective to only import the XX_B (back adjusted continuous) symbols when using CSI.

CSI provides a wide array of options for how to model the rolls from one contract to the next when stitching individual contracts together into a continuous contract.

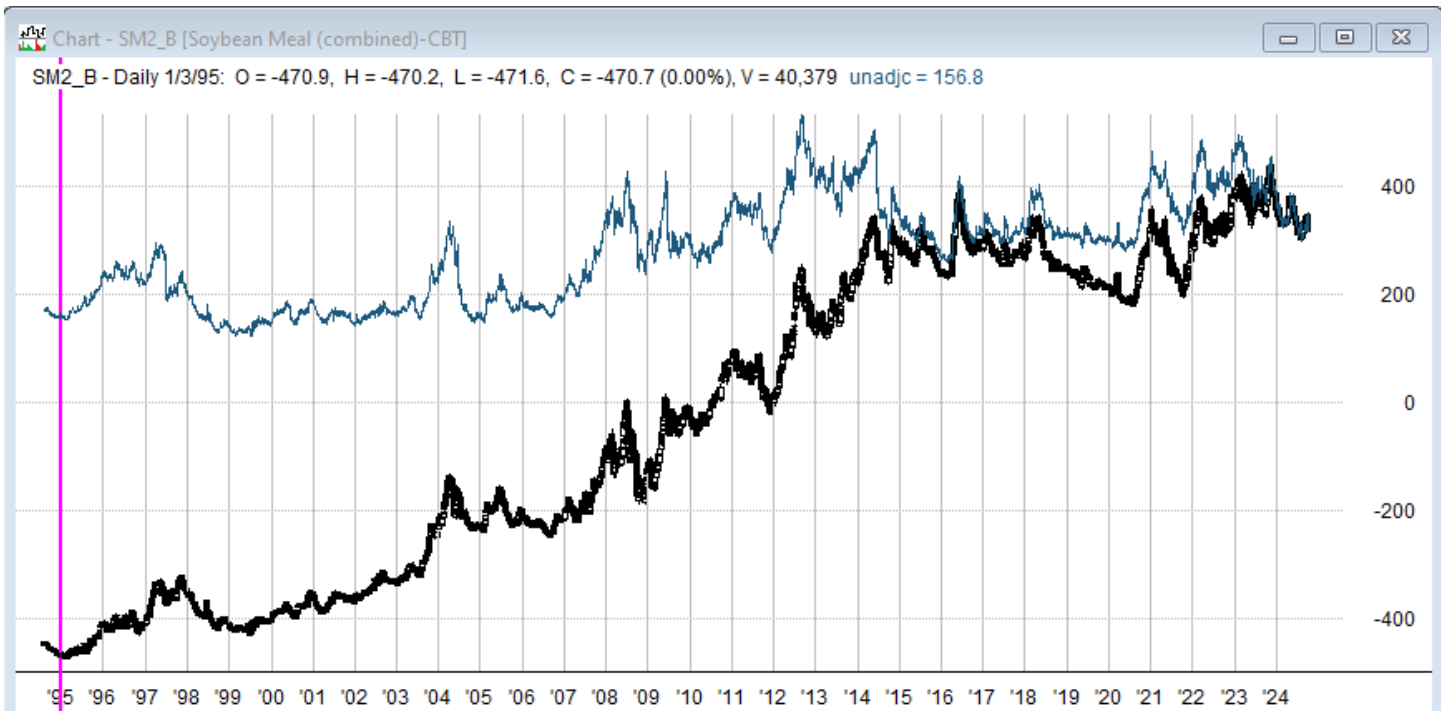
Among other choices, CSI supports proportional (ratio) adjustment and forward adjustment. These are NOT recommended for use with RealTest. Choose one of the difference-based back-adjusted approaches. The default settings in CSI's portfolio manager are a great place to start. See **CSI Futures Import** for further details.

As shown below for the VX futures continuous contract series, the bar data is back adjusted and also includes the unadjusted closes in **Extra** and the contract expiry dates in **InfoExpiry**:



RealTest has no problem handling the negative prices that will sometimes result from difference-based back adjustment. The key point is that the daily price movement, in points, remains unaffected by the adjustments (except on roll dates, when the price change from the prior day accurately reflects the difference between the two contracts involved in the roll).

Here's a market that, opposite to VX, tends to be in backwardation rather than contango:



Here is a trade example that illustrates this point:

The key stats in the trade record are **Points** and **Profit**. In contrast to those for stocks and ETFs, futures strategies generally do not use any type of calculations involving *percent change of price* in their trading rules or stats, nor *multiples of price* in their position sizing. Instead, these calculations focus on *points of movement,* **PointValue**, and point-based indicators such as **ATR**.

See below for the entire script of the strategy that made this trade.

When CSI continuous contract data is used with RealTest, roll costs can optionally be calculated using the **RollCost** strategy formula. RealTest knows when CSI has implemented each roll, because each continuous contract data series includes the expiry date of the current contract as of each bar. When *RollCost* is specified, the formula is evaluated on each roll date and the amount that it returns is subtracted from the net profit of the trade.

Add a **T.RollCosts** column to your *Trades.rts* to see the total roll costs for each trade in your trade lists. Add a **S.RollCosts** graph to your Graphs.rts to see the daily total roll costs applied to trades exiting that day. Add a formula with *Sum(S.RollCosts, S.Number)* to your Results.rts to show that figure as a column in your results windows.

When using continuous contract data, the symbols shown in your trade list will be the continuous contract symbol, e.g. ES_B for E-mini S&P 500.

If you enable the **Test Details Log**, you will see the entry, exit, and roll transactions logged using the individual contract symbols that would have been traded:

```
4/9/21    open    tf_csi_long    SM2_B    roll    4    from SM2_2021K to SM2_2021N, cost is $56.00
4/9/21    open    tf_csi_long    W2_B     roll    3    from W2_2021K to W2_2021N, cost is $37.00
4/9/21    open    tf_csi_long    SB2_B    roll    7    from SB2_2021J to SB2_2021M, cost is $72.80
```

When you generate orders, they will also use the correct individual contract symbols. Text orders will use symbols like the ones in the log output above.

When generating CSV orders, you can use the example **exchangemap_csi.csv** file as your **ExchangeMap**, and the example **ibfutures_csi.csv** file as your **SymChangeList**:

```
OrderSettings:
    ExchangeMap:    Examples\exchangemap_csi.csv
    SymChangeList:  Examples\ibfutures_csi.csv
    OrdersMode:     OrderClerk
```

This will ensure that your generated orders can actually be placed and executed in your Interactive Brokers account:

Date	TradeId	Strategy	Side	IsExit	OrderId	ParentId	OcaId	RTSym	Symbol	Expiry	SecType	Currency	Exchange	Action	Quantity	OrderType
2024-06-21	0	tf_csi_long	1	0	1			RA_B	ZAR	20240916	FUT	USD	SMART/CME	BUY	3	MKT
2024-06-21	0	tf_csi_short	-1	0	2			BR5_B	BRE	20240628	FUT	USD	SMART/CME	SELL	5	MKT
2024-06-21	0	tf_csi_short	-1	0	3			JY_B	JPY	20240916	FUT	USD	SMART/CME	SELL	2	MKT
2024-06-21	0	tf_csi_short	-1	0	4			MW2_B	MW2	20240913	FUT	USD	SMART/CME	SELL	1	MKT
2024-06-21	0	tf_csi_short	-1	0	5			O2_B	ZO	20241213	FUT	USD	SMART/CBOT	SELL	2	MKT
2024-06-21	0	tf_csi_short	-1	0	6			S2_B	ZS	20241114	FUT	USD	SMART/CBOT	SELL	1	MKT

(If you use a different broker that also accepts CSV order lists, you can modify your **OrdersTemplate** and the above two files as needed.)

The example script **futures_trend_follow_csi.rts** shows a complete long/short trend following strategy implementation:

```
Active Script - C:\RealTest\SCRIPTS\Examples\futures_trend_follow_csi.rts*

Notes:
    a futures long/short trend-following example using CSI back-adjusted continuous contract data

Import:
    DataSource:     CSIFutures
    DataPath:       c:\UA\Files\RealTest {"CSI"} // change to your CSI futures portfolio location
    StartDate:      Earliest
    EndDate:        Latest
    SaveAs:         csi_futures.rtd

Settings:
    DataFile:       csi_futures.rtd
    StartDate:      Earliest
    EndDate:        6/20/24
    AccountSize:    1e6             // $1M
    CashIntPct:     2               // interest paid on cash not in use as margin collateral

Parameters:
    VolMin:         500             // minimum recent volume
    BreakLen:       250             // breakout lookback length
    PctRisk:        0.5             // percent of capital to risk per trade
    StopATRs:       7               // trailing stop width, in ATRs

Data:
    N:              ATR(20)
    NDlr:           N * PointValue
    NStopPts:       N * StopATRs
    NStopDlr:       NStopPts * PointValue
    Liquid:         InList("CSI") and Lowest(V, 5) > VolMin
    BreakHigh:      Highest(C, BreakLen)
    BreakLow:       Lowest(C, BreakLen)
    RiskPts:        Min(BreakHigh - BreakLow, NStopPts) * PointValue
    LongStop:       Max(BreakLow, C - NStopPts)
    ShortStop:      Min(BreakHigh, C + NStopPts)

Template: common
    Quantity:       PctRisk / 100 * S.Alloc / RiskPts // risk-based position size
    Commission:     2 * Contracts   // $2/contract each way
    Slippage:       2 * TickSize    // 2 ticks each way
    RollCost:       4 * Contracts + 4 * TickSize * PointValue // commission and slippage for each full roll
    FuturesMargin:  10 * NDlr       // estimated margin requirement per contract
    MinFreeCash:    0.5 * S.Alloc   // allow up to half NLV to be used as margin, else skip entries
    ExitStopTime:   NextOpen        // define "stop" as closing below the stop price (comment out for live stops)

Strategy: tf_csi_long
    Using:          common
    Side:           Long
    EntrySetup:     Liquid and C = BreakHigh
    ExitStop:       Highest(LongStop, BarsHeld + 1) // Trailing Stop

Strategy: tf_csi_short
    Using:          common
    Side:           Short
    EntrySetup:     Liquid and C = BreakLow
    ExitStop:       Lowest(ShortStop, BarsHeld + 1) // Trailing Stop
```

14. RealTest Script Language

RealTest scripts describe strategies and analyses without traditional programming. These topics cover the language, its sections, and every syntax element.

- **Language Basics** - section and item syntax, and the intelligent script editor
- **Script Sections** - the outer sections that make up a script
- **Formula Syntax** - the expression language used throughout
- **Syntax Element Categories** - every keyword grouped by category
- **Syntax Element Details** - the full reference for each keyword
- **Bar Offsets** - referring to earlier bars
- **Date Constants** - writing dates in formulas
- **Other Constants** - other built-in constants
- **Symbol References** - referring to specific symbols
- **String Values** - text values in scripts
- **User-Defined Item Names** - naming your own items
- **Item Name Overrides** - overriding reserved names
- **Script Comments** - commenting out text
- **Conditional Comments** - including or excluding blocks by definition
- **File Path Syntax** - how paths are written and resolved
- **Output Format Specification** - controlling column formatting
- **New Script Template** - the starting point for a new script
- **Combining Scripts** - including one script in another
- **Default Scripts** - automatically included section files

14.1. Language Basics

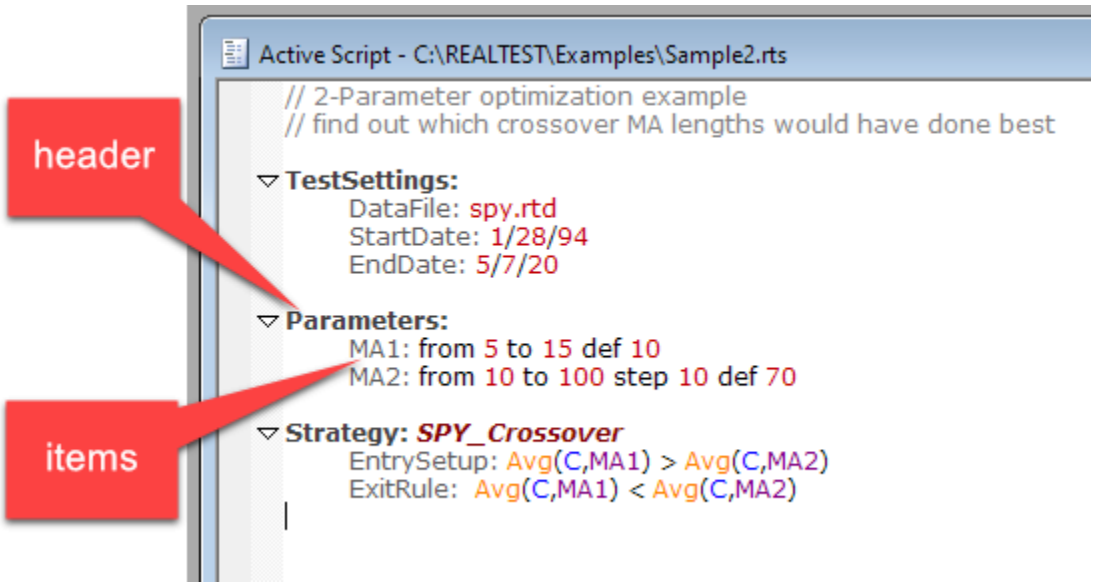
Writing RealTest scripts does not require any programming in a traditional sense.

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.



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 the 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.

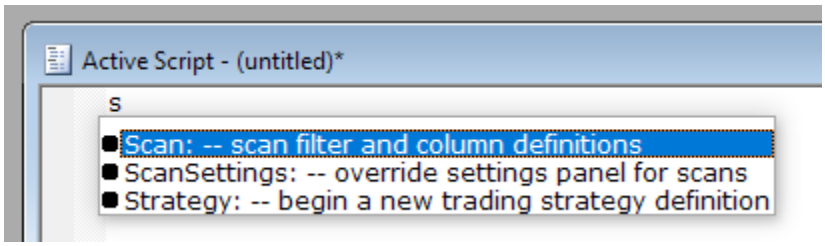
For readability, it is recommended to give each section header and each item its own line, and to indent the items one tab stop. Among other benefits, this permits "folding" in the script editor, and it also facilitates context determination for the context-sensitive help.

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.

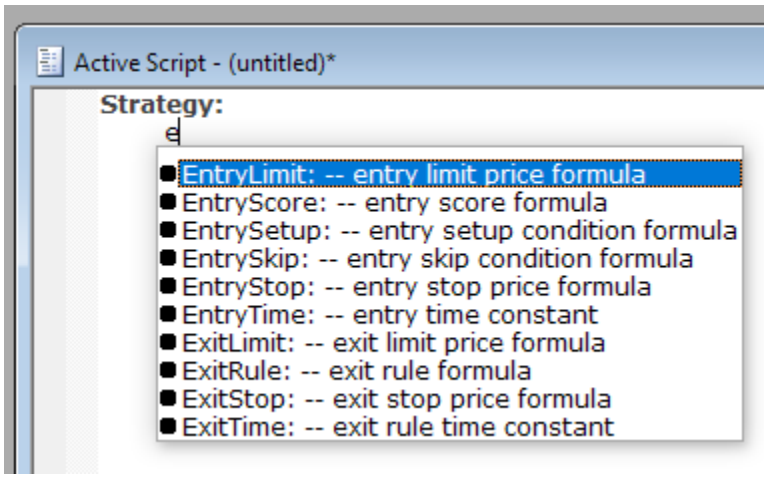
Try opening a new blank script and typing the letter 's':



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:



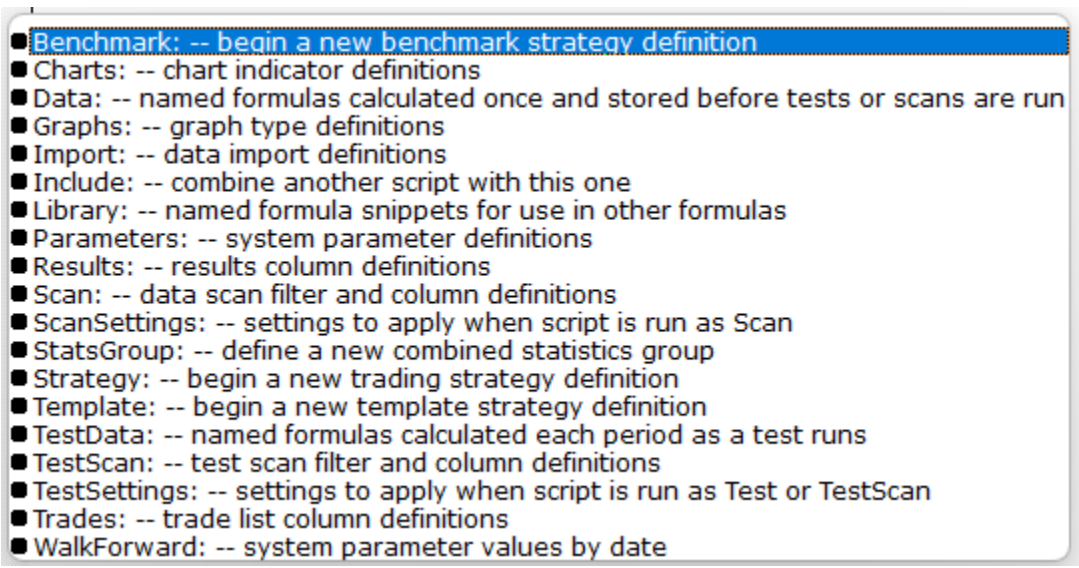
Select *EntrySetup, then type 'c' to begin your formula:*

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 too.

14.2. 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.

14.2.1. 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.

14.2.2. Data Section

The Data section calculates named data columns that every formula can reference. These topics explain how it works and its special item types.

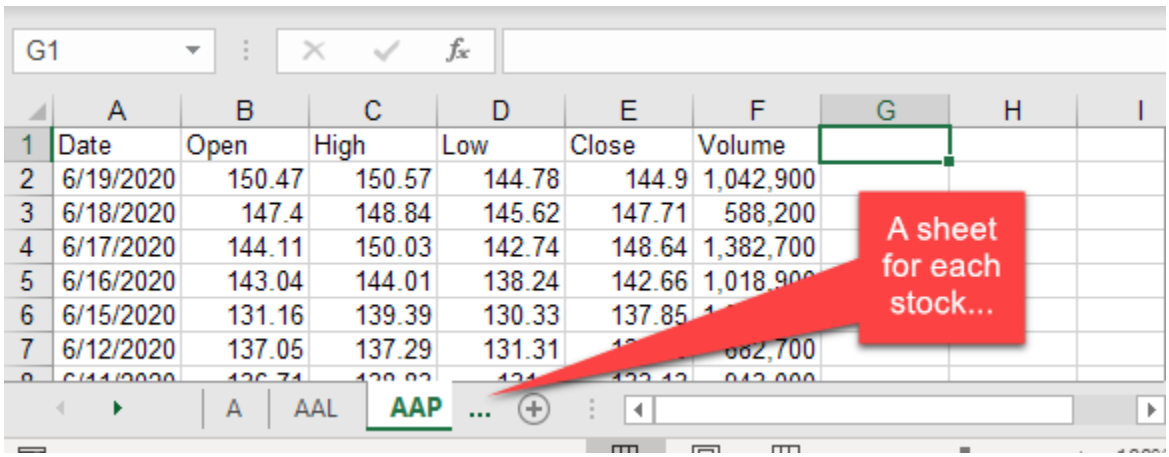
- **How the Data Section Works** - how data columns are calculated and stored for every symbol
- **One-Pass Data Formulas** - formulas evaluated in a single pass
- **Self-Referential Items** - items that refer to their own prior values
- **Bar-Size-Specific Items** - items tied to a particular bar size
- **DataValueFile Items** - reading data columns from an external CSV

14.2.2.1. How the Data Section Works

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.

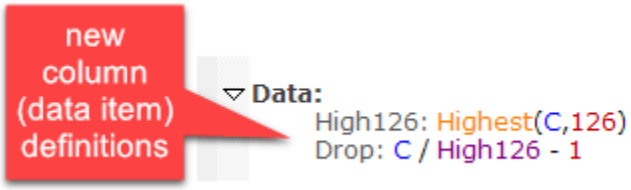
Say you have just imported daily OHLCV bars for all the stocks in the S&P 500 index, as shown in **Tutorial 3**.

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:

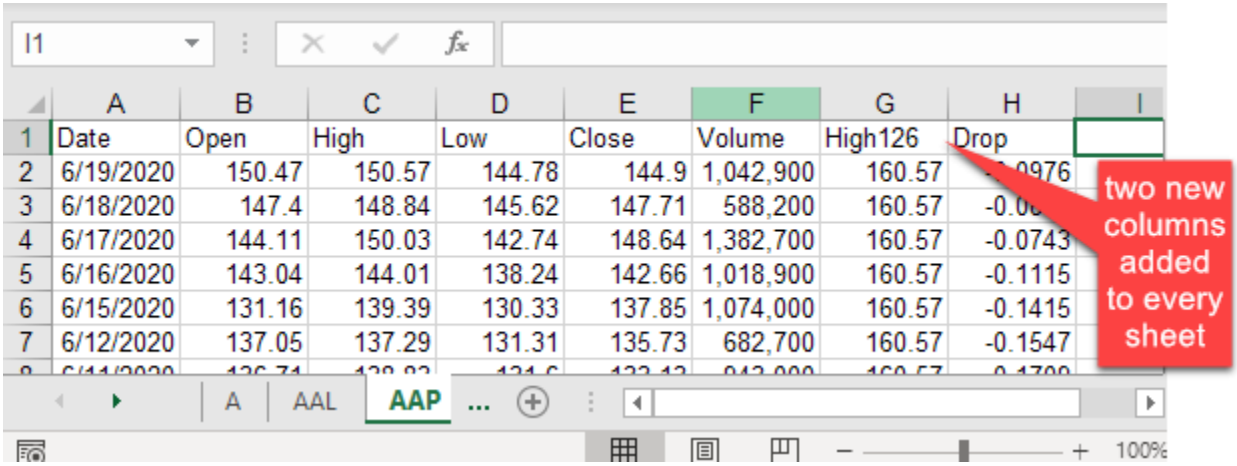


Now when you run a script that includes a Data section, here's what happens:

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.



Now the imaginary workbook looks like this:



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.

The stored values do consume memory in proportion to symbols × bars × items, so a very large script on a machine with limited RAM may not fit. If that happens, see **Data On Demand** for an option that runs the same script in a fraction of the memory.

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.

14.2.2.2. 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!

14.2.2.3. 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*(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.

14.2.2.4. 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.

For example, if the settings bar size is *Daily* and the script includes a *Weekly* strategy, the Data items used by that strategy must be explicitly specified as weekly bar arrays.

To begin a group of data items with a specific bar size, you can add a **BarSize** statement within the Data Section itself.

For example, the following would calculate and store a 20-week moving average in a daily bar size

script:

```
Settings:
    BarSize:      Daily

Data:
    BarSize:      Weekly
    Avg20:        Avg(C, 20)
```

All items defined after a *BarSize* statement and before the next *BarSize* statement will be calculated and stored in that bar size.

Another way to designate an individual data item as having a specific bar size is to simply prefix its name with that *BarSize* name.

This example would also calculate and store a 20-week moving average in a daily bar size script:

```
Settings:
    BarSize:      Daily

Data:
    weeklyAvg20:  Avg(C, 20)
```

Whether a BarSize statement is added to create a group of items with that bar size or a bar size name prefix is used for an individual item, such items are calculated and stored using bars of the designated size.

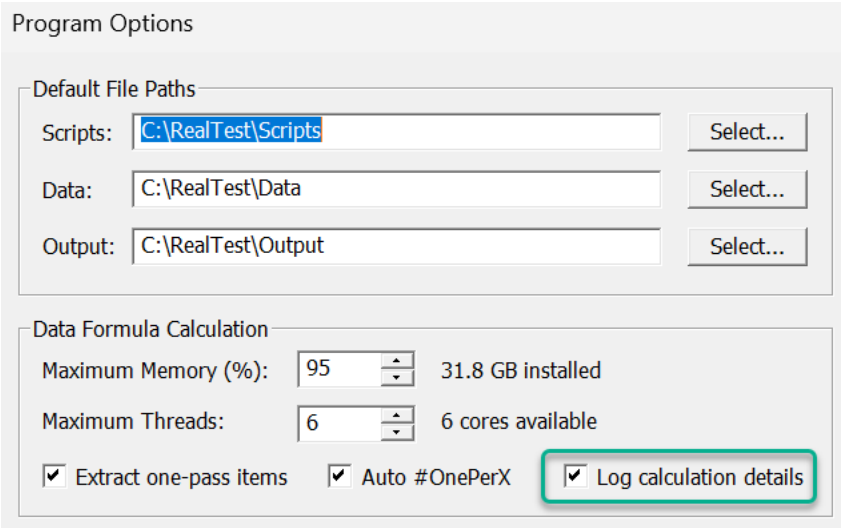
It is also possible to calculate a data formula using bars of a different size than it is stored with.

This example, like the one above, calculates a 20-week moving average:

```
Settings:
    BarSize:      Daily

Data:
    externAvg20:  Extern(~Weekly, Avg(C, 20))
```

Since the Settings bar size is daily, the weekly values are expanded into a daily array in memory. This takes more memory than the other approaches described above but may result in faster script run times if that item is being used by a daily bar strategy.

To see all the internal details about your Data items including their bar size and calculation speed, check this box in **Program Options**:



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 non-daily 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 Data BarSize or name prefix method is preferable when calculating non-daily 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, choose the other-barsize Data item style to match the context from which it will be referenced.

14.2.2.5. DataValueFile Items

Most **Data Section** items in most scripts will specify formulas to calculate the data values.

DataValueFile items are a special exception to this rule. Rather than calculating their values, these items read them from a CSV file.

This feature can be thought of as a way to add any number of additional columns with data from external sources to your OHLCV bar data after having already imported it.

To use a DataValueFile your Data section needs two special elements:

1. One or more DataValueFile statements to specify the path(s) to the CSV file(s) containing your custom data.
2. One or more named Data items (columns) with **#DataValueFile** as their definition in place of a formula

The CSV files are formatted as follows:

Date	Symbol	myData1	myData2	myData3	etc.
2020-01-02	AAPL	123.45	234.56	345.67	
2020-01-03	AAPL	234.56	345.67	456.78	
2020-01-04	AAPL	345.67	456.78	567.89	
etc.					

The first row is a header row with Date and Symbol as the first two columns, in that order. Any number of additional columns can follow these first two. The names of these columns must match the #DataValueFile item names in your Data section.

Inside the file, columns must be delimited with commas, not semicolons etc. Dates can be provided in various **formats**. Numbers must use period as the decimal separator. Values must not be quoted. All data must be numeric -- RealTest does not allow strings in Data section arrays.

A #DataValueFile Data section item can optionally specify **#DVFAlign**, **#Fill** and/or **#Pad** to control how non-market dates and missing values are handled.

The Data section for the above CSV file would look like:

```
Data:
   DataValueFile: c:\my_folder\my_file.csv
   myData1: #DataValueFile
   myData2: #DataValueFile
   myData3: #DataValueFile
```

Note that these special Data item definitions do not include anything other than the above special tags -- there are no values or formulas.

RealTest will read and process every DataValueFile CSV file declared in the script at the following times:

- whenever the entire Data section needs to be recalculated
- whenever any #DataValueFile items have been added, modified, removed, renamed or relocated
- whenever the last modified time of the file is newer than the last time the section was recalculated

When the DataValueFile CSV file(s) is(are) reloaded, every #DataValueFile item's data array is first re-initialized to all nans.

Then, for each row of each CSV file:

1. The symbol is looked up in the currently-loaded RTD file. If not found, log an error and proceed to the next row.
2. The date is looked up in the symbol's date list. If outside the entire date range of the symbol, proceed to next row without logging anything. If between two existing dates and #DVFAlign was specified, change it to the later of those two dates, otherwise log the incorrect date and proceed to the next row.
3. For each subsequent column with a header name that matches a #DataValueFile item name, read the item value and store it in the appropriate cell of the data array.
4. At the end of the file processing, if #DVFPad or #DVFFill were specified, loop over all values of all symbols. If Fill was specified (which takes precedence over Pad) then change each missing value (nan) to the last found value, else if Pad was specified then change each nan to 0.

At this point the data from the file(s) are in memory just as if they had been calculated by Data item formulas, and can be referenced by any formula in the script in all the usual ways.

14.2.2.6. Data On Demand

Every item in a script's **Data Section** is normally calculated once, up front, and stored in memory -- one value per bar per symbol. References during a test are then simple array reads, and an item that several formulas share is never computed twice. The cost is storage: memory use grows with symbols × bars × items, and a multi-market, multi-system script over a long history can need more memory than the machine has.

The **Data on demand** option (on the Calculation tab of the **Program Options** dialog, default off) trades calculation time for memory. When it is enabled, eligible Data items are not stored at all; each reference evaluates the item's formula on the fly, the same way a **Library** formula would be. A test that needed tens of gigabytes of Data storage can run in almost none -- it just runs more slowly, since values are recomputed at every bar where they are referenced.

The option applies automatically to every eligible item, so the same script runs unchanged on a large-memory and a small-memory machine. There is no need to maintain a separate low-memory copy of a script with its Data formulas moved to Library.

Which Items Stay Stored

Some kinds of Data items only work as stored arrays, and RealTest keeps them stored even when the option is enabled:

- Cross-sectional (breadth) items -- **breadth tags** such as *#Rank*, *#PctRank*, *#Count*, *#Avg* and *#Sum* are computed across all symbols, date by date, and must be stored. (Their input formulas can still be other, on-demand items.)
- **Self-referential items** -- cumulative formulas that refer to their own prior values, e.g. *cum: cum[1] + ...*
- **DataValueFile items** -- their values come from an external file, not a formula.
- **#OnePerDate** and **#OnePerSym** items -- these already store only one value per date or per symbol, so there is little to save.
- Items whose formulas contain *Random()* -- the stored draws must persist so that every reference sees the same value.
- DLL data-calc items, and any items of the same bar size defined before one -- the user DLL is handed pointers to the stored arrays.
- Items built on exponential smoothing or other recursive calculations -- the EMA-family functions (*EMA*, *ESD*, *AEMA*, *AESD*, *DEMA*, *TEMA*, *KAMA*, *RsiF*, *SarF*) and the indicators built on them (*RSI*, *RRSI*, *CRSI*, *ATR*, *ADX*, *PDI*, *MDI*, *MACD*, *MACDS*, *MACDH*, *KBTop*, *KBBot*, *SuperTrend* and the Ehlers filters). Their stored calculation carries exact running state across the whole history, which a per-reference evaluation could only approximate, so they stay stored to guarantee

identical results (see below).

Everything else -- simple moving averages and other windowed calculations, comparisons and combinations, items that reference other Data items, items at a different bar size than the test -- is evaluated on demand.

Results Match Stored Calculation

An on-demand item returns the same value a stored read would have returned, so test results do not change when the option is toggled. The recursive EMA-family items listed above stay stored for precisely this reason: evaluated on the fly they would use the bounded lookback that the EMA precision setting defines, which is normally indistinguishable from the full-history calculation but can diverge sharply in special cases (after a long one-directional run, an on-the-fly *RSI* can read 0 where the full-history value is near 100).

The memory cost of this exclusion is smaller than it may appear. When an EMA-family function is part of a larger formula, the "Extract one-pass items" optimization normally splits it out into its own hidden stored item first, and only that piece stays stored -- the containing item can still evaluate on demand. (An EMA-family call written directly in a strategy formula, where it is not extracted, continues to use the bounded-lookback evaluation it always has; that behavior is independent of this option.)

Performance Notes

- The cost of a reference is proportional to the formula's lookback window (e.g. *Avg(C, 200)* scans 200 bars each time it is referenced). Items that reference other on-demand items multiply this cost, so keep deeply nested chains in mind if a test is unexpectedly slow.
- Breadth items evaluate their input formulas once per symbol per date during calculation, so scripts that rank complex on-demand factors with *#Rank* feel the cost most.
- A per-thread cache absorbs repeated references to the same item at the same bar (multiple strategies, entry plus exit formulas), which is the most common pattern.
- With "Log calculation details" (same Calculation tab) enabled, on-demand items are marked *** in the data calculation log, so you can see exactly which items were stored and which were not.

The Out-of-Memory Check

Independently of this option, RealTest computes the projected size of the Data section before allocating it and compares it to the amount of memory that can currently be committed (available RAM plus page file). A projection that merely exceeds physical RAM is allowed to run - the operating system uses the page file as needed, which is slow but works.

If the projection exceeds what can currently be committed, the Maximum Memory setting (on the General tab of the **Program Options** dialog) decides what happens next. When projected memory use stays within that limit, RealTest asks whether to continue, since a system-managed Windows page file can grow to make up the difference; in **command line mode**, a warning goes to the log and the run continues. When projected use would exceed the limit, the run stops immediately with a message reporting how many gigabytes the calculated data items require and how many can be committed, and suggesting that you enable Data on demand if it is not already on, reduce the amount of data loaded, or raise Maximum Memory. In command line mode, the message goes to the error log and the run exits with the memory error code (4).

14.2.3. TestData and StratData Sections

TestData and **StratData** are specialized variations of the **Data** section.

To summarize their similarities and differences:

	Data Section	TestData Section	StratData Section
When is it calculated?	before any test starts, for all bars of all stocks	during each test, for each stock, at the end of each test date	during each test, for eac strategy at the end of ea 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-lookback indicators are avoided	like *TestData* but with or one column per strategy so speed is not a conce
What can its formulas reference?	bar data, Parameters, previously calculated Data items (including itself), or Library items that only reference these things	everything that Data formulas can reference, anything that Strategy formulas can reference, and other TestData items (including itself)	nothing symbol-specific just the summary-leve stats (S.xxx) of the strat
Cross-sectional (breadth) support?	yes	yes	yes
How is it stored?	8 bytes per item times number of stocks times number of bars	ditto	8 bytes per item times number of strategies tim number of test periods

TestData and *StratData* formulas are evaluated during a test at the end of processing for each test date (*BarSize* period).

TestData formulas are evaluated for 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.

StratData formulas are evaluated for the current day of every strategy being tested (including *Benchmarks*, *StatsGroups* and *Combined*), with the value then stored in the *StratData* item's memory array.

As with *Data* items, *TestData* and *StratData* 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 and *StratData* items can also **refer to themselves** in the same way that *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
```

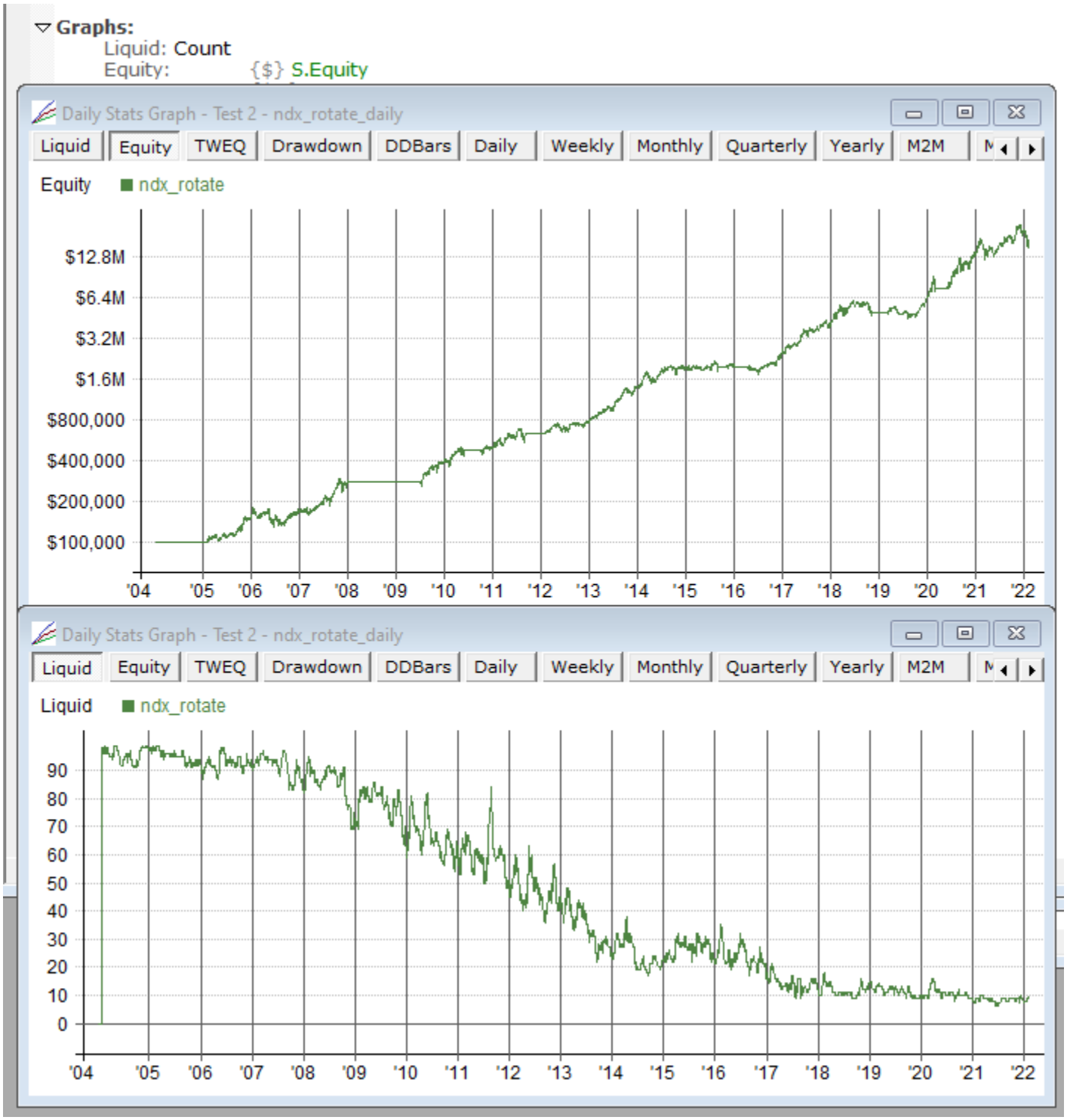
Note that in the above example, *S.Equity* is referenced directly. This works because there's only one strategy in the script.

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" ...

The example above requires *TestData* because it needs to calculate a rank factor formula for each symbol.

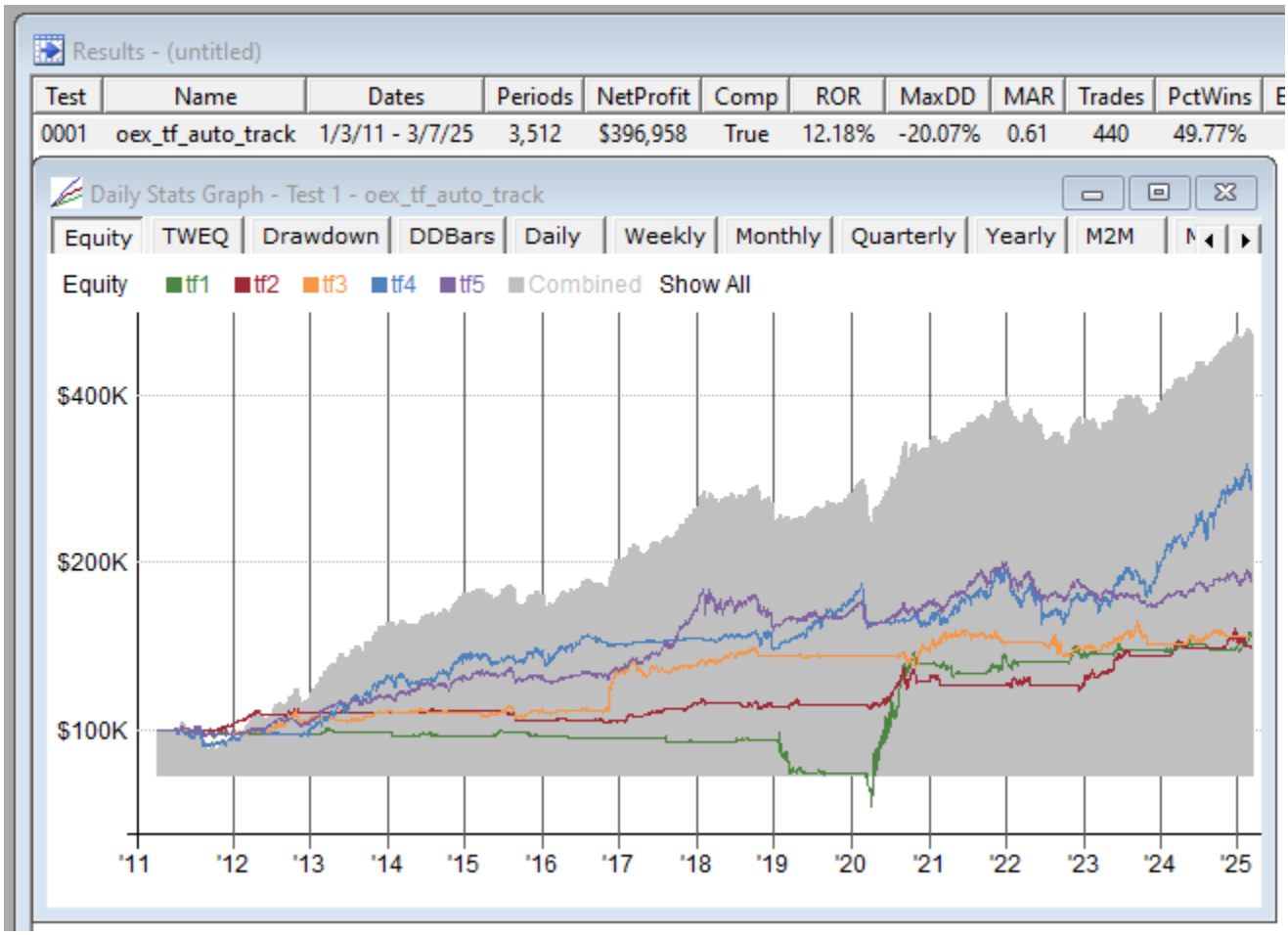
Use *StratData* when you want to calculate and store values that do not require individual symbol data, and/or when you'd like to be able to apply cross-sectional calculations to your set of strategies and/or benchmarks.

The **oex_tf_auto_track.rts** example script demonstrates this:

```
StratData:
        // tracker benchmark rank factor
        factor:         S.Equity / MA(S.Equity, 50)
        // tracker benchmark rank number
        srank:          #Rank  if(IsTracker, factor, nan) // nan excludes strategies themselves

Template: oex_tf
        Side:           Long
        Tracker:        Hidden // change to Visible to see their equity graphs and stats
        // skip strategy entries if its corresponding tracker benchmark has too high (poor) performance rank
        SetupSkip:      if(IsTracker, 0, Extern(@tracker, srank > maxrank))
```

This part of the script implements the auto-tracker mechanism. The script includes five versions of the same trend-following strategy that each use the above template. The "maxrank" parameter loops from 1 to 5 to test how many of the top-ranked strategies to allow new entries for.

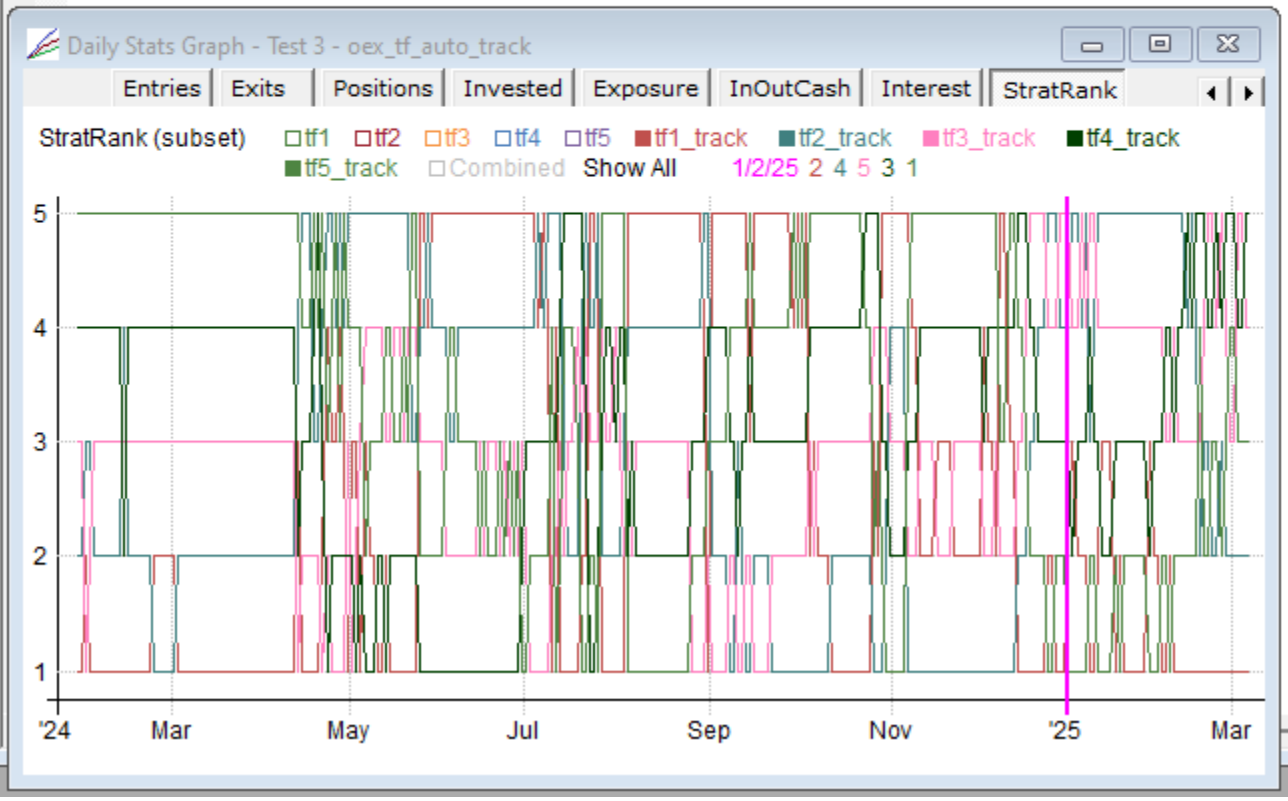
These are the results with maxrank = 2:



Note that this trades all five strategies, but on any given date will only add new positions in the top two.

To see more detail about how this works, change the **Tracker** setting to *Visible*, rerun the script, add a graph to show the rankings, and open it:

```
Template: oex_tf
    Side:          Long
    Tracker:       Visible // change to Visible to see their equity graphs and stats
    // skip strategy entries if its corresponding tracker benchmark has too high (poor) performance rank
    SetupSkip:     if(IsTracker, 0, Extern(@tracker, srank > maxrank))

Include: c:\RealTest\Graphs.rts
Graphs:
    StratRank:     srank
```



Now we see the five auto-tracker benchmarks and the changes in their rank numbers.

(This also shows how to add your own graphs at the end of the default ones by including the default script first.)

14.2.4. 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-Filter 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 SaveScanAs, 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.

14.2.5. 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 when running a **Scan**.

Use **TestSettings** for settings to apply when running a **Test** or **TestScan**.

Use **OptimizeSettings** for settings to apply specifically when running an **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 setting can also be restated within or across settings sections, in which case the last statement evaluated wins. Restating a file output setting (SaveTradesAs, SaveStatsAs, ResultsFile, etc.) with a blank value clears it, which is mainly useful in scripts that **Include** another script that specifies those settings.

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:

Setting	Type	Default
AccountSize	dollars	settings panel
AccountType	CashNextDay, CashSameDay, Margin	Margin
AllowSameName	True or False	False
AlwaysImport	True or False	False
BarSize	Daily, Weekly, Monthly, Quarterly, Yearly	settings panel
CashIntPct	number	0
CommCurrency	USD, EUR, GBP, CAD, AUD, etc.	none
Currency	USD, EUR, GBP, CAD, AUD, etc.	none
DataFile	folder path and file name	settings panel
DateDisplay	Excel-style date definition string	M/d/yy
DateInput	DMY or MDY	MDY
DateSym	symbol (no leading $)	none
DaysPerYear	number	252
DividendsToCash	True or False	False
EndDate	date	settings panel
EndOfTestExits	True or False	True
ExchangeMap	folder path and file name	none
HolidayList	folder path and file name	none
KeepTrades	see topic for choices	settings panel
LogDetail	see topic for choices	All
MarginIntPct	number	none
NumBars	number	settings panel
OptAnchor	None, Start, or End	None
OptByStrategy	True or False	False
OptBySymbol	True or False	False

OptClearResults	True or False	True
OptContinueResults	True or False	False
OptimizeMode	Combinatorial, Sequential, Genetic, Random, Defaults	Combinatorial
OptIterations	number	1
OptKeepBest	number	0
OptKeepByDate	True or False	False
OptKeepBySymbol	True or False	False
OptKeepNoTrades	True or False	False
OptMultiSaveAs	True or False	False
OptMutatePct	number	25
OptNoDialog	True or False	False
OptNoStats	True or False	False
OptScoreCol	Results column name	none
OptShards	number (2 to 32)	0
OptShowResults	Live, AtEnd, or None	Live
OptSortResults	True or False	False
OptSummaryLog	True or False	False
OptTestInterval	number	1
OptTestLength	number	1
OptTimeUnit	None, Years, Months, Weeks, Days	None
OptWalkForward	True or False	False
OrderClerkFolder	folder path	none
OrdersComment	string or formula	none
OrdersFile	folder path and file name	none
OrdersFreeCash	folder path and file name	none
OrdersLiveData	True or False	False
OrdersMktAsLmtPct	number	none
OrdersMode	see topic for choices	Text
OrdersNetLiq	folder path and file name	none
OrdersTemplate	folder path and file name	none
OrdersUseQtyIn	True or False	False
RandomSeed	number or expression of Parameters	none
ReportContent	see topic for choices	settings panel
ReportsFolder	folder path	none
ResultsByYear	ROR, MaxDD, Trades (comma-separated) or None	None
ResultsFile	folder path and file name	none
RiskFreeRateSym	symbol (no leading $)	none
SaveLogAs	folder path and file name	none
SavePositionsAs	folder path and file name	none
SaveScanAs	folder path and file name	none
SaveScanChartsMax	number	no limit
SaveScanChartsTo	folder path	none
SaveStatsAs	folder path and file name	none
SaveStatsInclude	Strategies, Benchmarks, StatsGroups, Combined	all are included
SaveTestListAs	folder path and file name	none

SaveTradeChartsMax	number	no limit
SaveTradeChartsTo	folder path	none
SaveTradesAs	folder path and file name	none
SaveTradesType	Compact or Full	Full
ScanNoDefCols	True or False	False
ScanNoHeader	True or False	False
ScanNoWindow	True or False	False
ShortLoanIntPct	number	0
ShortPrcdsIntPct	number	0
SkipTestIf	formula (references parameters)	none
StartDate	date	settings panel
SymChangeList	folder path and file name	none
TestName	string	settings panel or script name
TestOutput	see topic for choices	settings panel
TestScanAllDates	True or False	False
TestScanPositions	True or False	False
UseAvailableBars	True or False	False

14.2.6. 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 **clenow_stocks_on_move.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 built-in variables **Arg1 - Arg9** (or **Ref1 - Ref9** to receive an argument by reference).

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).

Arguments referenced as Arg1-Arg9 are passed "by value". In the above example, the current values of H, L, C would be passed to the AvgOfThree function; because each is a single number, the function cannot, for example, access Arg3[1] to get the prior close.

To receive an argument "by reference" instead -- re-evaluated in the caller's context each time it is used, so that Ref3[1] and the use of arguments in multi-bar functions both work as expected -- reference it as Ref1-Ref9 rather than Arg1-Arg9. See **Arg1 - Arg9 / Ref1 - Ref9**.

14.2.7. Strategy Section

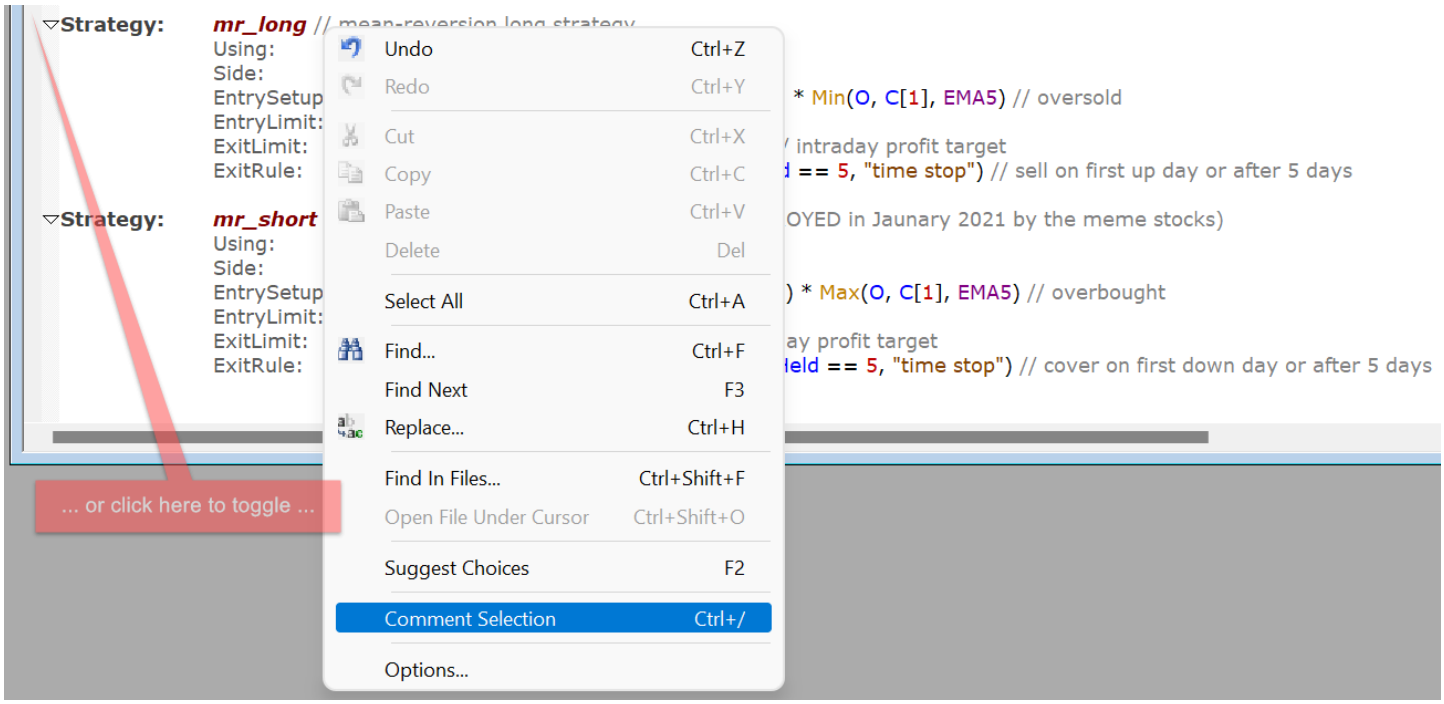
A Strategy section defines the rules of one trading strategy. These topics describe how strategy sections work and the elements they contain.

- **Strategy Basics** - using one or more strategies and commenting them in or out
- **Strategy Names** - naming your strategies
- **Special Strategy Types** - benchmark, tracker, and other special strategies
- **Strategy Elements** - the entry, exit, sizing, and other elements
- **Strategy Element Value Types and Defaults** - value types and defaults for each element

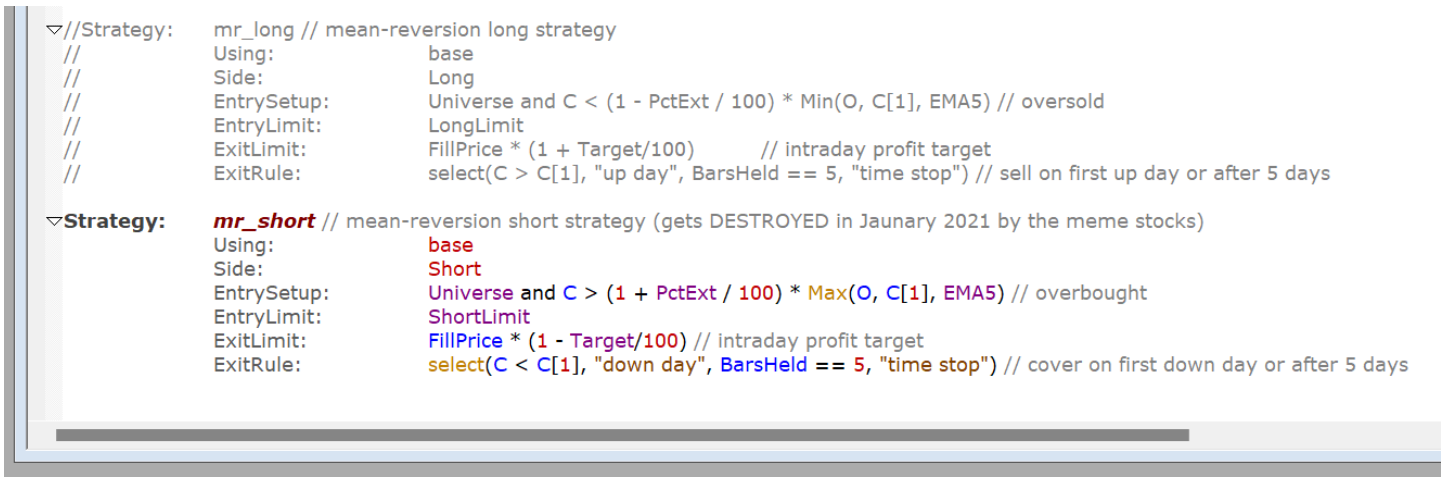
14.2.7.1. Strategy Basics

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**.



… resulting in …



Do the same in reverse to remove the comments and include the strategy again.

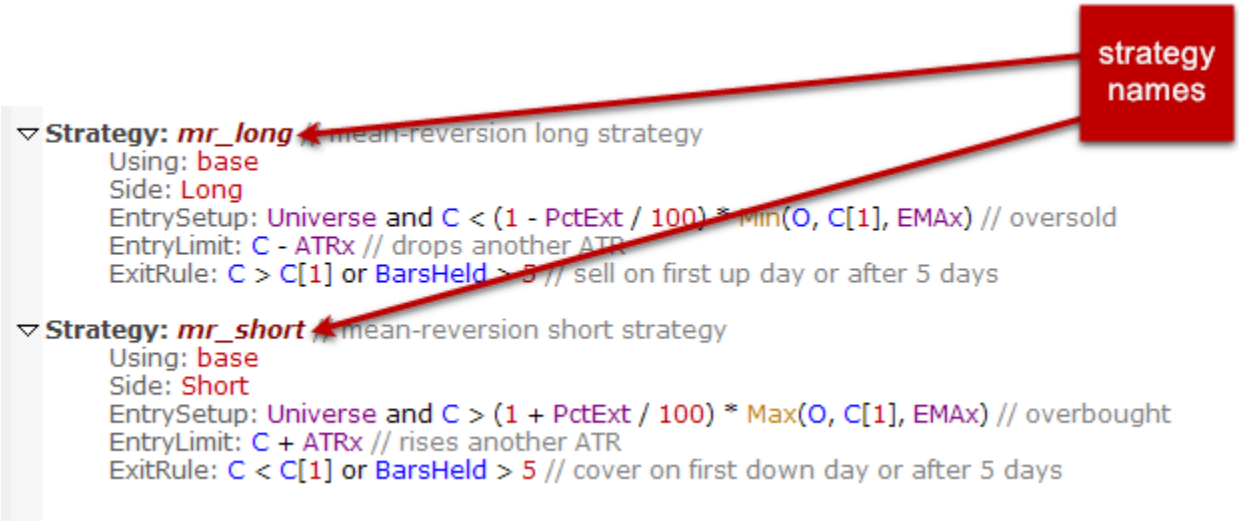
Another way to toggle comment in/out is to click in the left margin adjacent to the line or section to be toggled, as shown above.

14.2.7.2. Strategy Names

Each strategy must be given a name (up to 80 characters in length).

Type the name after the "Strategy:" marker as shown below.

The name you provide will be used in all references to the strategy in graphs, trade lists, etc.



14.2.7.3. 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.

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. Note that a benchmark position whose symbol stops printing bars before the test ends is held to the end of the test rather than being flattened, so that the benchmark remains referenceable - see **End-of-Data Exits**.

By default, each benchmark processes its entry setups in its own entry phase, in declaration order, ahead of all strategies - so a strategy that references a benchmark always sees the benchmark's final same-bar order state. A benchmark can instead be given a later **EntryPhase** than the strategies, so that it reacts to their final orders.

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**, **MaxInvested**, **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.

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%.

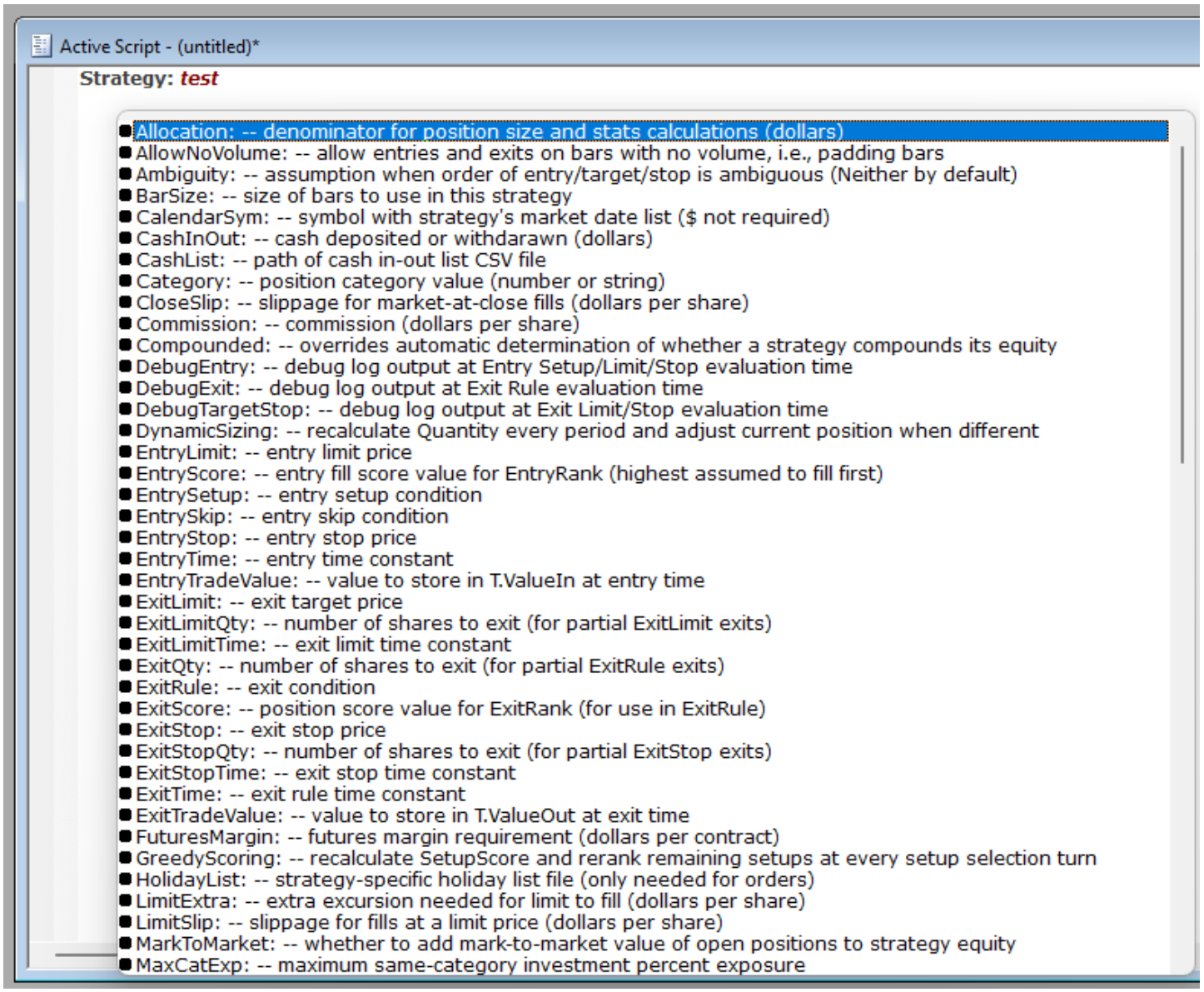
Similarly, these constraints in a *StatsGroup* ensure that setups will be skipped if entering those positions would exceed any them.

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.2.7.4. 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):



(there are many more if you scroll down, or just start typing to get a more focused list)

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.

14.2.7.5. 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
Allocation	formula	dollars	full account size
AllowNoVolume	constant	*True* or *False*	False
Ambiguity	constant	*Default,Stop,Target,Neither*	*Neither*
BarSize	constant	*Daily,Weekly,Monthly,Quarterly,Yearly*	*Daily*

CalendarSym	constant	symbol (no leading $)	none
CashInOut	formula	dollars	0
CashList	file	file path	none
Category	formula	number	0 (none)
CloseSlip	formula	dollars per share	0 (use slippage)
Collateral / FuturesMargin	formula	dollars	**InfoMargin**
Commission	formula	dollars	0 (none)
Compounded	constant	*True* or *False*	inferred from Quantity
DebugEntry	formula	string or number	none
DebugExit	formula	string or number	none
DebugTargetStop	formula	string or number	none
DynamicSizing	constant	True or False	False
EntryLimit	formula	price	0 (none)
EntryScore	formula	number (highest first)	alphabetical by symbol
EntrySetup	formula	condition	0 (false)
EntrySkip	formula	condition	0 (false)
EntryStop	formula	price	0 (none)
EntryTime	constant	logical time of day*	*NextOpen*
EntryTradeValue	formula	number	0
ExitLimit	formula	price	0 (none)
ExitQty	formula	shares	entire position
ExitLimitTime	constant	logical time of day*	*Intraday*
ExitLimitQty	formula	shares	entire position
ExitRule	formula	condition	0 (false)
ExitStop	formula	price	0 (none)
ExitStopQty	formula	shares	entire position
ExitStopTime	constant	logical time of day*	*Intraday*
ExitTime	constant	logical time of day*	*NextOpen*
ExitTradeValue	formula	number	0
GreedyScoring	constant	*True* or *False*	False
HolidayList	file	file path	none
LimitExtra	formula	dollars per share	0 (none)
LimitSlip	formula	dollars per share	0 (use Slippage)
MarkToMarket	constant	*True* or *False*	True
MaxCatExp	formula	percent	unlimited
MaxCatInv	formula	dollars	unlimited
MaxEntries	formula	count	unlimited
MaxExposure	formula	percent	unlimited
MaxInvested	formula	dollars	unlimited
MaxLongExp	formula	percent	unlimited
MaxLongInv	formula	dollars	unlimited
MaxNetExp	formula	percent	unlimited
MaxNetInv	formula	percent	unlimited
MaxNewExp	formula	percent	unlimited
MaxNewInv	formula	dollars	unlimited
MaxNewPos	formula	count	unlimited
MaxPerTurn	formula	count	1
MaxPositions	formula	count	unlimited

MaxSameCat	formula	count	unlimited
MaxSameSym	formula	count	1 for strategy, unlimited for statsgroup or combined
MaxSetups	formula	count	unlimited
MaxShortExp	formula	percent	unlimited
MaxShortInv	formula	dollars	unlimited
MaxSymExp	formula	percent	unlimited
MaxSymInv	formula	dollars	unlimited
MinFreeCash	formula	dollars	unlimited
MinNetExp	formula	percent	unlimited
MinNetInv	formula	dollars	unlimited
OpenSlip	formula	dollars per share	0 (use slippage)
OrdersMktAsLmtPct	formula	number	0
OrderNote	formula	number or string	none
OrdersFile	constant	file path	none
PriceRound	formula	number	tick size
QtyFinal	formula	shares or contracts	Quantity forumla value
QtyPrice	constant	*OrderPrice* or *FillPrice*	OrderPrice in default mode
QtyRound	formula	number	1
QtyType	constant	*Shares, Value, Percent*	*Shares*
Quantity	formula	shares or contracts	account size / entry price
Reduce	constant	*True* or *False*	False
RollCost	formula	dollars	0
SetupScore	formula	number (highest first)	alphabetical by symbol
SetupSkip	formula	condition	0 (false)
Side	constant	*Long,Short,Both*	*Both*
Slippage	formula	dollars per share	0 (none)
StopSlip	formula	dollars per share	0 (use Slippage)
StrategyScore	formula	number (highest first)	script sequence
TLAdjusted	constant	*True* or *False*	False
TLDateFmt	constant	MDY or DMY	program options setting
TLDelim	constant	*Comma, Semicolon, Tab*	
TLFields	constant	ordered list of field names	none
TLEndDate	constant or date	*0*, *Latest*, or a date	False
TLNumFmt	constant	*Point* or *Comma*	
TLStratName	constant	string	current strategy name
TLTimeShift	constant	number of hours	0
TradeList	file	file path	no trade list
Using	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.

14.2.8. Parameters Section

Parameters or optimization variables are defined in their own section of the script.

They can be given any name, which can then be referenced from any formula in the script.

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, e.g. "1, 3, 17, 44"
- A comma-separated list can also end with a "def" clause, e.g. "10, 20, 50 def 20", making that value the default without reordering the list

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
```

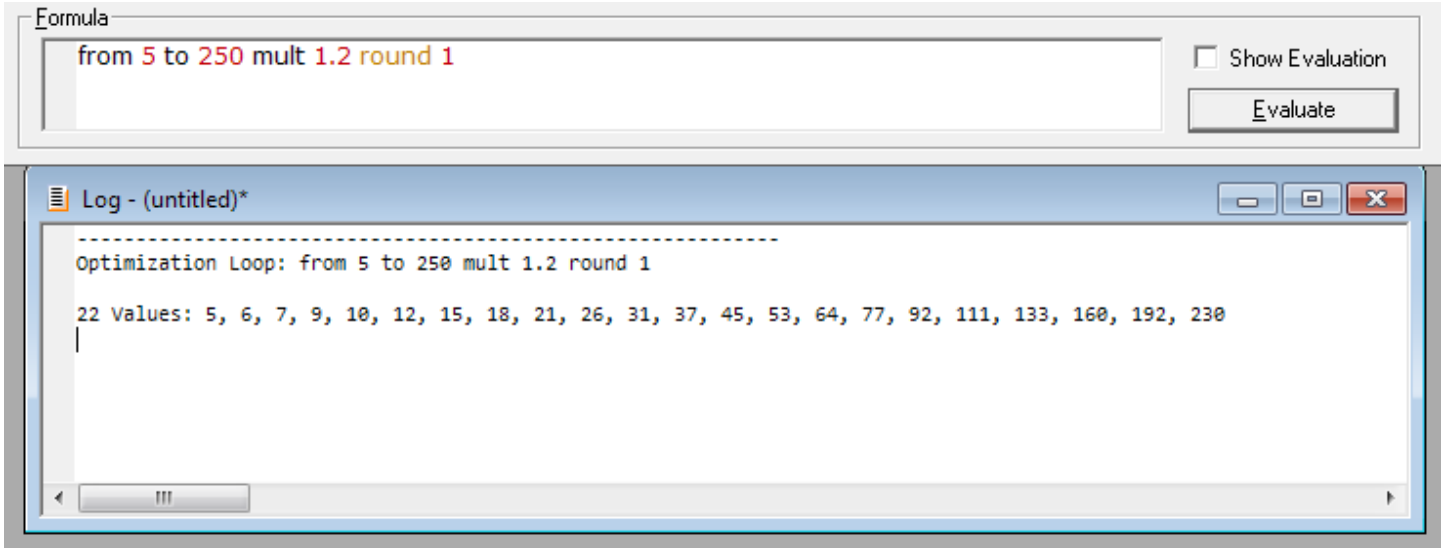
If no default ("def") is specified, then the default value will be the first number in a list or loop.

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.



To specify within the Parameters section which of your parameters should be selected by default in the **RealTest Optimizer**, add {X} to the definition of each parameter that you want to be selected.

Two commands can edit this section for you: the Optimizer dialog's **Save to Script** button adds or removes the {X} markers to match the dialog's parameter check boxes, and the Results menu's **Save Parameters to Script** command (see the **Results Menu**) sets "def" clauses from a selected test's parameter values.

A third command, the Results menu's **Apply Parameters and Data**, edits no script: it sets a selected test's parameter values in memory and recalculates dependent Data items with them, so that trade lists, chart indicators, graphs, plots and the Debug panel show that test's context. The next run or apply reverts to the script's own values.

Note that parameters (like Data and Library items) remain in memory after a run so that charts and the Debug panel can continue to evaluate them. They are only cleared when the next script run declares its own Parameters section. This means that if you run a script with no Parameters section of its own, the previous script's parameter names remain defined, and formulas that reference them will still resolve. To explicitly clear them, declare an empty Parameters section.

14.2.9. Results Section

The Results Section is where the columns to display in any **Results Window** are defined. Each item becomes one column.

Here is the default set of Results window columns and their definitions:

```
Active Script - C:\RealTest\Scripts\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 trade to end of test
                                          // use S.Number alone if you prefer "Periods" (and stats) to include dates with no tr
          NetProfit:      {$0}            S.Equity - S.StartEquity - S.CashInOut
//        Dividends:      {$0}            Sum(S.Dividends, Periods)
//        Comms:          {$0}            Sum(S.Comms, Periods)
//        Interest:       {$0}            Sum(S.Interest, Periods)
//        Cash:           {$0}            S.CashInOut // already cumulative
//        Fees:           {$0}            S.FeesInOut // already cumulative
//        NetFX:          {$0}            Sum(S.NetFx, Periods)
//        Rolls:          {$0}            Sum(S.RollCosts, Periods)

          _years:                         Days(S.Date[Periods-1], S.Date) / 365.25
          _CAR:                           (S.TWEQ/S.TWEQ[Periods-1])^(1/_years)-1
          _AAR:                           (((S.Equity - S.CashInOut) / S.StartEquity) - 1) / _years
          Comp:           {~}             S.Compounded // shows whether stats were compounded

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

          ROR:            {%2|}           iif(Comp, _CAR, _AAR)
          MaxDD:          {%2|}           -S.MaxDDPct
          MAR:            {#2}            ROR / -MAXDD // return over maxdd aka Mar Ratio

//        AvgMar:         {#2}            ROR / Avg(S.DDPct, Periods) // return over avgdd

          // Calmar Ratio (3-year MAR) calculation (assumes compounded)
//        _Per3Y:         3 * S.BPY
//        _Car3Y:                         ((S.Equity - S.CashInOut) / (S.Equity - S.CashInOut)[_Per3Y])^(1/3) - 1
//        _MDD3Y:                         Highest(S.DDPct, _Per3Y)
//        Calmar:         {#2}            _Car3Y / _MDD3Y

          // misc. other stats
//        UI:             {#}             Sqr(SumSq(100 * S.DDPct, Periods) / Periods) // Ulcer Index
//        MAE:            {%2}            Lowest(S.MAE/S.Alloc[1],Periods-1) // Max Adverse Excursion
//        MFE:            {%2}            Highest(S.MFE/S.Alloc[1],Periods-1) // Max Favorable Excursion

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

          // position-size-based trade-level summary stats
          Expectancy:     {%-2}           Sum(S.TradePct,Periods) / Exits
```

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. A Results section in the script being run replaces the default

set entirely.

Examples of some columns that you might want to add are provided as **comments** in this default script (rows beginning with //).

How and when the formulas are evaluated

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. Each formula is evaluated separately for every strategy in the test as well as for the test as a whole, which is how one definition fills both the test row and its per-strategy rows.

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. If a test ended up with 6,894 stats records, the number of the last record - and therefore the value of S.Number when the Results formulas run - is 6,894.

Similarly, for stats where we want the total count of something for the entire test, we use the **Sum** function over that many periods. A stats array can also be indexed backward from the end of the test: *S.Date[Periods-1]* in the default script is the date of the first period counted.

Results formulas can optionally reference **Trade Statistics Functions**, though calculating these can be slow for tests with very high trade counts.

Items that refer to other items

Any item can refer to any item defined earlier in the same section, by name and without regard to case. In the default script *Periods* is defined once and then used by nearly every formula below it, and *MAR* is simply *ROR / -MaxDD*.

An item whose name begins with an underscore is calculated but not displayed. Use these for intermediate results that you do not want as columns of their own - *_years*, *_CAR*, *_wins* and *_losses* in the default script are examples.

Formatting and column headings

A **format specification comment** can be included in any item. If no format is specified, the item will use default number formatting. Both the short-glyph form (*{$0}*, *{%2}*, *{#2}*, *{~}*) and the named-attribute form (*{format: dollar, places: 0}*) are accepted, and the two forms can be mixed freely from item to item.

A quoted string in the format block overrides the column heading. The default script uses *Exits: {#|"Trades"}* because *Trades* is a reserved word and cannot be an item name, but it makes a better heading. The named form of the same thing is *{name: "Trades"}*. Everything that refers to a column by name - sorting, CSV headers, **OptScoreCol** - uses the overridden heading, not the item name.

The vertical bar

The vertical bar *{|}* in a Results item format specification (*{line: Bars}* in the named form) singles that item out for two extra uses:

- it is calculated after each day of the test and displayed in the **status bar** as the test is running
- it is included in the Optimization Summary written to the log at the end of an optimization run - one row per marked column giving its minimum, maximum, median, average and standard deviation across all of the tests

Because a marked item is re-evaluated on every day of every test, it is worth keeping the number of them small.

Where else these columns are used

- They are the sortable headings of the **Results Window** and the fields written to results CSV output by **ResultsFile**.
- **OptScoreCol** names the one Results column used to rank an optimization. It must match a displayed column, so an underscore item cannot be used as the score.
- Any displayed column can be chosen as the metric plotted in **Optimization Results Graphs**,

including the parameter sensitivity histogram.

One exception to "edit and re-apply": tests saved with **OptNoStats** keep no stats arrays, only the column values as they stood when the test was saved. Reopening such results matches the saved values to the current section by column name, so a renamed or newly added column shows as blank rather than being recalculated.

To apply Results section formulas to all currently open Results windows, press F4 or click Apply.

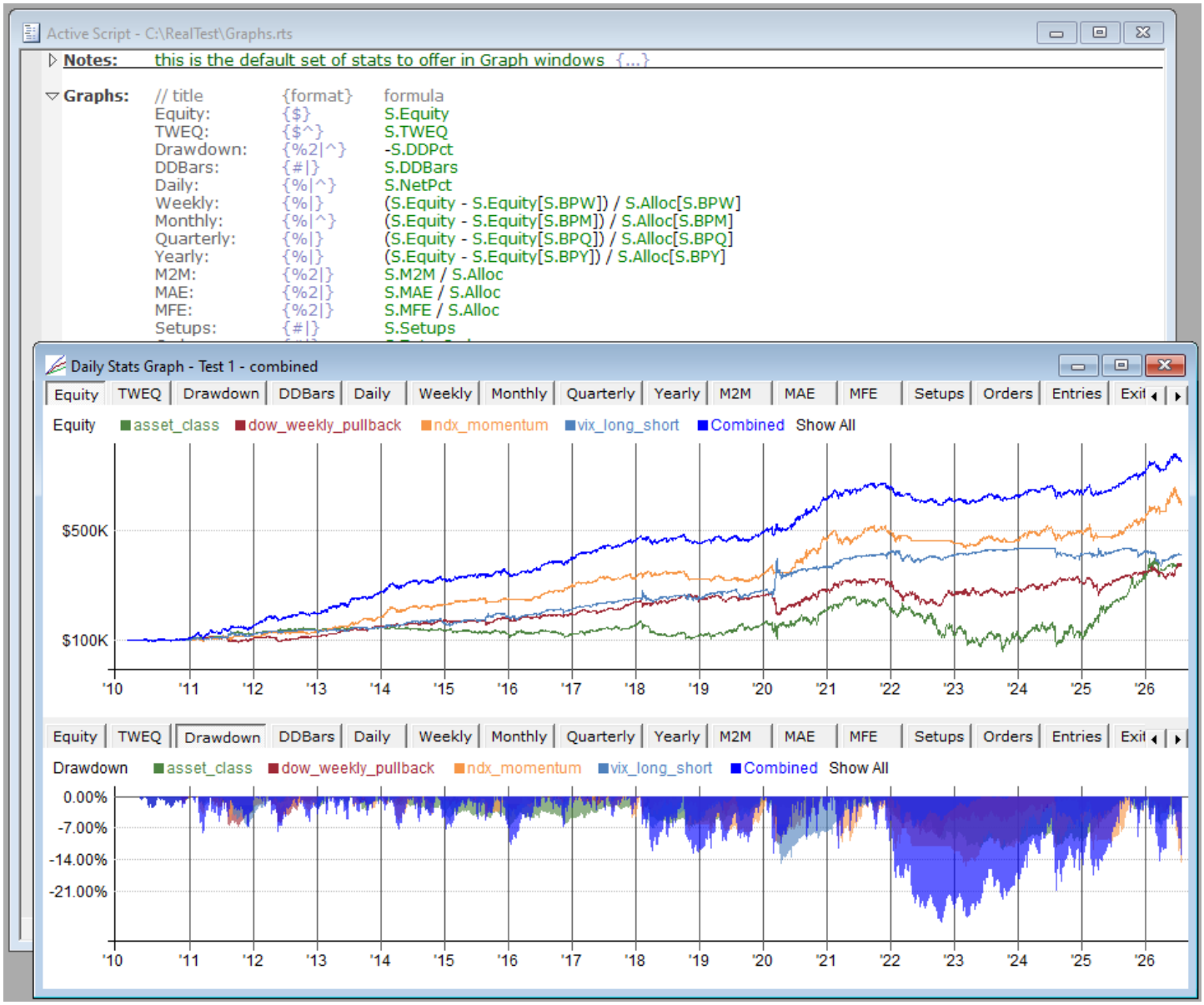
See also **Default Scripts** and **Output Format Specification**.

14.2.10. 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.

Every graph is drawn once for each strategy in the test and once for the test as a whole, which is why one formula produces the whole legend of lines shown above. The color of each line comes from the strategy palette in the graph options dialog, unless the **Strategy** (or Template, Benchmark, StatsGroup or Combined) section header carries a *{color: ...}* comment of its own.

The formulas refer to the **Test Statistics Arrays**, which can be indexed backward from the date being plotted - *S.Equity[S.BPM]* is equity as of one month ago, which is how the Monthly graph in the default script computes a rolling monthly return. Any item can also refer to any item defined earlier in the same section, by name, and those references can be indexed the same way.

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.

Format Specifications

A **format specification comment** can be included in any item and has two purposes: it selects how the series is drawn, and it sets the number format used for the Y axis and for the value readout when the cursor is on a date. If no format is specified, the item is graphed as a line using default number formatting.

The named-attribute form is the recommended way to write these. The script editor pops up a list of the attributes that apply while the cursor is inside the *{ }* block. These are the attributes that apply to Graphs items:

- *line: Line | Bars* - a continuous line (the default) or a histogram. (Graphs accept *Dashed* and *Dotted* but treat them as *Line*.)
- *base: n* - with *line: Bars*, the value the bars are drawn from (0 unless you say otherwise)
- *format:* and *places:* - the number format for the axis and the cursor readout
- *name: "label"* - a tab and legend label that differs from the item name
- *hat: 1* - include this graph in **Test Summary Reports**

The older short-glyph codes still work and can be mixed freely with named-attribute items in the same section. The default *graphs.rts* shown above is written entirely in that form: the vertical bar *{|}* is the same as *{line: Bars}* (and *{|3}* is that with *base: 3*), the accent *{^}* is the same as *{hat: 1}*, and *{%2}* is the same as *{format: percent, places: 2}*. So *{%2|^}* on the Drawdown graph could equally be written *{format: percent, places: 2, line: Bars, hat: 1}*.

Note that *color:*, *pane:* and *width:* have no effect here - they belong to **Charts** items. A graph's colors are per strategy, not per graph.

Why the choice of line type matters

Two graph window features are available only for items drawn as bars:

- The Sort command re-plots the bars in descending order instead of by date, turning the graph into a distribution of the stat over the whole test. It is disabled for line graphs, whose x axis has to remain the date axis.
- Bar fills honor transparency, so overlapping strategies remain readable. Give a strategy a color with an alpha component (*#RRGGBBAA* or *ARGB(a,r,g,b)*), or set Bar Graph Opacity in the graph options dialog to make every bar graph translucent at once. Lines always draw opaque.

That is why the default script draws equity curves as lines and per-period returns, drawdowns and counts as bars.

Test Summary Reports

The special format code *{^}* (or *{hat: 1}*) is used to specify which graphs are included in **Test Summary Reports**.

A graph marked {^2} or {^3} is stacked beneath the preceding {^} graph in the report instead of getting its own image: the group becomes one combined image with a shared date axis, split 60/40 for two graphs or 50/30/20 for three, and named for the first graph in the group. A slot number less than or equal to its predecessor's starts a new image, so scripts that use plain {^} on every graph render exactly as before.

To apply Graphs section contents to all currently open Graph windows, press F4 or click Apply.

See also **Default Scripts** and **Output Format Specification**.

14.2.11. Trades Section

The *Trades Section* is where the additional columns to display in any **Trade List Window are defined.**

The first 13 columns of every trade list - Trade, Strategy, Symbol, Side, DateIn, TimeIn, QtyIn, PriceIn, DateOut, TimeOut, QtyOut, PriceOut and Reason - are always present and are not customizable. Everything to the right of Reason comes from this section.

Here is an example of a script (the default trades.RTS) which adds some custom columns:

```
Active Script - C:\RealTest\Scripts\Trades.rts
Notes:   this is the default set of additional columns for Trades windows {...}

Trades:  // item:        {format}  formula
//       Filter:                   Symbol=$AAPL                  // uncomment and edit this line, then press F4 to
//       Sort:                     -DateOut, Symbol              // uncomment and edit this line to specify default
         Bars:           {#}       T.Bars
         PctGain:        {%-2}     T.NetPct                      // gain as fraction of position size
//       PctGainAlloc:   {%-3}     T.Fraction * T.NetPct         // gain as fraction of account size at entry time
//       PctGainPrice:   {%-2}     T.Points / T.PriceIn          // simple price gain
         Profit:         {$-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 * T.PtVal
//       Comms:          {$2}      T.CommIn + T.CommOut
//       Slips:          {$2}      T.SlipIn + T.SlipOut
//       Rolls:          {$2}      T.RollCosts * T.FxOut
         Dividends:      {$-2}     T.Div
```

Applying this script causes the new columns to be shown in all open trade windows. Here is the resulting trade list, with the fixed columns through Reason followed by the custom ones defined above:

Trade List - Test 3 - 3,875 Trades

Trade	Strategy	Symbol	Side	DateIn	TimeIn	QtyIn	PriceIn	DateOut	TimeOut	QtyOut	PriceOut	Reason	Bars	PctGain	Profit	PctMFE	PctMAE	Fraction	Size
03251	shorts	NYT	Short	2/6/25	open	2,488	49.65	2/21/25	open	2,488	48.94	time stop	10	1.41%	$1,741.60	3.20%	-2.52%	9.98%	$123,529
03252	longs	VKTX	Long	2/20/25	open	2,942	32.11	2/21/25	intraday	2,942	34.73	exit limit	1	8.13%	$7,678.62	8.16%	-3.05%	7.39%	$94,468
03253	shorts	PEGA	Short	2/14/25	open	925	85.30	2/21/25	intraday	925	78.46	exit limit	4	8.01%	$6,317.74	8.02%	-1.50%	6.30%	$78,902
03254	shorts	TTD	Short	2/14/25	open	516	82.55	2/21/25	intraday	516	73.67	exit limit	4	10.75%	$4,576.92	10.76%	-1.28%	3.38%	$42,596
03255	shorts	FOUR	Short	2/20/25	open	864	104.00	2/21/25	intraday	864	95.67	exit limit	1	8.00%	$7,188.48	8.01%	-1.01%	6.97%	$89,856
03256	shorts	WEX	Short	2/7/25	open	604	152.00	2/24/25	open	604	150.49	time stop	10	0.99%	$906.00	3.93%	-4.88%	7.37%	$91,808
03257	momentum	AFRM	Long	2/10/25	open	331	75.59	2/24/25	open	331	68.91	exit rule	9	-8.85%	($2,214.40)	9.18%	-9.66%	2.00%	$25,020
03258	momentum	AS	Long	2/18/25	open	822	31.00	2/24/25	open	822	29.06	exit rule	4	-6.29%	($1,602.90)	1.61%	-9.13%	2.00%	$25,482
03259	shorts	GDDY	Short	2/18/25	open	617	182.01	2/24/25	intraday	617	172.38	exit limit	4	5.29%	$5,935.53	5.29%	-0.23%	8.82%	$112,300
03260	longs	VMI	Long	2/19/25	open	305	367.22	2/25/25	open	305	343.50	price stop	4	-6.46%	($7,237.66)	2.01%	-7.09%	8.94%	$112,002
03261	longs	ROK	Long	2/11/25	open	512	304.99	2/26/25	open	512	290.78	price stop	10	-4.23%	($6,609.92)	0.66%	-5.70%	12.31%	$156,155
03262	shorts	CE	Short	2/20/25	open	1,145	54.83	2/28/25	intraday	1,145	49.60	exit limit	6	9.47%	$5,942.54	9.54%	-0.80%	4.89%	$62,780
03263	shorts	TFX	Short	2/28/25	open	517	135.45	2/28/25	intraday	517	130.84	exit limit	0	3.40%	$2,378.19	3.40%	-1.82%	5.89%	$70,028
03264	shorts	CGNX	Short	2/14/25	open	2,748	33.96	3/3/25	open	2,748	32.86	time stop	10	2.97%	$2,775.48	5.36%	-0.00%	7.44%	$93,322
03265	momentum	CVNA	Long	7/8/24	open	138	129.06	3/3/25	open	138	234.50	exit rule	163	81.69%	$14,548.72	126.90%	-8.18%	1.99%	$17,810
03266	momentum	MSTR	Long	9/16/24	open	130	137.02	3/3/25	open	130	293.15	exit rule	114	113.94%	$20,294.90	296.29%	-6.21%	2.00%	$17,813
03267	momentum	VST	Long	12/30...	open	166	136.05	3/3/25	open	166	133.66	exit rule	41	-1.77%	($398.74)	46.89%	-8.82%	2.00%	$22,584
03268	longs	PRGO	Long	3/3/25	open	3,699	29.58	3/3/25	intraday	3,699	30.37	exit limit	0	2.64%	$2,885.21	2.67%	-2.92%	8.66%	$109,416
03269	longs	BFAM	Long	2/18/25	open	1,261	128.40	3/4/25	open	1,261	127.73	time stop	10	-0.53%	($857.49)	1.64%	-3.90%	12.76%	$161,912
03270	longs	OKTA	Long	3/5/25	open	882	108.64	3/5/25	intraday	882	114.99	exit limit	0	5.84%	$5,591.88	5.84%	-1.16%	7.81%	$95,820
03271	shorts	PLNT	Short	2/26/25	open	1,353	90.34	3/6/25	open	1,353	94.58	price stop	6	-4.70%	($5,750.26)	1.75%	-7.63%	9.90%	$122,230
03272	shorts	AAP	Short	2/27/25	open	1,727	37.76	3/6/25	intraday	1,727	33.69	exit limit	5	10.75%	$7,011.61	10.78%	-2.46%	5.26%	$65,212

The context of a Trades formula

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).

The exception is an item tagged with the *{factor}* attribute: it is evaluated as of the trade's **setup** bar instead, with no offset needed, so that its column shows exactly the value a **Factor Scan** uses. To see both readings of the same indicator, define it twice - once tagged, once not.

Items that refer to other items

Any item can refer to any item defined earlier in the same section, by name and without regard to case. An item whose name begins with an underscore is calculated but not displayed, which lets a column be built up in readable steps. The default script uses that to derive the excursion columns: *_UP* and *_DN* measure the favorable and adverse moves, *_MFE* and *_MAE* pick the right one according to *T.Side*, and only *PctMFE* and *PctMAE* become columns.

Formatting and column headings

A **format specification comment** can be included in any item. If no format is specified, the item will use default number formatting. Both the short-glyph form (*{$-2}*, *{%2}*, *{#}*) and the named-attribute form (*{format: dollar, places: 2, negred: 1}*) are accepted. The minus sign in *{$-2}* is what makes losing trades show in red in the screen shot above.

A quoted string in the format block (or *{name: "..."}*) overrides the column heading, which is the way to use a heading that would not be a legal item name.

Literal strings or string functions such as Format can also be used in custom trade list columns.

Filter and Sort

In addition to the above, there are two special Trades items that can optionally be added: **Filter** and **Sort**. Neither one produces a column, and neither name can be used as a column name.

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. Give a comma-separated list of column names, most significant first, each optionally prefixed with - for descending order - the default script's commented-out example is *-DateOut, Symbol*. The names may be any of the 13 fixed columns or any displayed column defined in this section; an unrecognized name is reported as a script error.

Trade list output files

The same set of columns defines the CSV written by **SaveTradesAs**, so adding a column here adds a field to that file. The exception is **SaveTradesType**: *Compact* writes only the fields needed for **imported trade list** playback and ignores this section.

To apply Trades section formulas to all currently open Trade List windows, press F4 or click Apply. To open the script behind a trade list, press F9 or use the context menu.

See also **Default Scripts** and **Output Format Specification**.

14.2.12. Charts Section

This script section defines the 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          formula              {format}
// Bollinger bands on chart
            BBupper:      BBTop(20, 2)         {color: purple}
            BBavg:        MA(C, 20)            {color: purple, line: dotted}
            BBlower:      BBBot(20, 2)         {color: purple}

// MACD Histogram in indicator pane 1
            MacdHist:     MACDH(12,26,9)       {pane: ind1, line: Bars, color: blue}

// RSI in indicator pane 2
            RSI14:        RSI(14)              {pane: ind2, color: orange}

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

Each item defined in the *Charts* section defines one data series. The item name becomes the name of the line in the chart's legend row, and the item formula is evaluated for every bar of the chart - at whatever bar size the chart is currently displaying - and then plotted.

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.

Any item can refer to any item defined earlier in the same section, by name, and the reference can be indexed like any other bar array - *_ma[1]* is the previous bar's value of *_ma*.

Chart formulas can use price bar elements and **Data section** elements, and - once a test has been run - **test statistics**, trade and current-position elements as well (see Trade Charts below).

The items defined above are plotted below.

To access the underlying formulas for any Charts, Results, Graphs or Trades window, press the F9 key or use the context menu. To apply the edited section to all currently open Chart windows, press F4 or click Apply in the Tool Bar.

Format Specifications

A **format specification comment** controls where and how each line is drawn. The named-attribute

form is the recommended way to write these - it is self-describing, and the script editor pops up a list of the attributes that apply while the cursor is inside the *{ }* block. These are the attributes that apply to Charts items:

- *pane: price* | *ind1* | *ind2* | *ind3* | *volume* - which pane to draw the line in (default *price*)
- *color:* - a named color, *#RRGGBB*, *RGB(r,g,b)*, or one of the transparency forms *#RRGGBBAA* / *ARGB(a,r,g,b)*
- *line: Line* | *Bars* | *Dashed* | *Dotted* - how the series is rendered
- *width: n* - pen width in pixels (1 to 20) for this line only, overriding the Line Width chart option; with *line: Bars* it sets each bar's half-width
- *base: n* - with *line: Bars*, the value the bars are drawn from (default 0)
- *format:* and *places:* - how the value is written in the legend
- *name: "label"* - a legend label that differs from the item name

The older short-glyph codes still work and can be mixed freely with named-attribute items in the same section. The accent *{^}* is the same as *{pane: ind1}* (*{^2}* and *{^3}* select the higher panes), and the vertical bar *{|}* is the same as *{pane: volume}*. Note that *{|}* always draws a line; only *line: Bars* draws actual bars.

If no format is specified at all, the item is plotted along with the price bars and uses their scale. If a number format is specified, it is used whenever the value of the item is displayed for a specific bar.

Panes

There are five possible destinations. *price* is the main pane with the price bars. *ind1*, *ind2* and *ind3* are indicator panes stacked above the price pane, with *ind1* closest to the price and *ind3* highest. *volume* is the pane below the price bars.

An indicator pane appears only if at least one item is assigned to it **and** the indicator pane display is turned on - press *i* or use the chart options dialog. The lower pane is likewise toggled with *v*. If no items are routed to the volume pane, it shows volume bars; assigning any item to it replaces them. (It is not currently possible to plot both indicators and volume bars in the lower pane.)

An item left in the price pane shares the price scale, so an indicator whose values are nowhere near the price will flatten the bars into a straight line. Put those in an indicator pane.

Turning a line off where you don't want it

A bar whose formula evaluates to *NaN* is simply not drawn, and the line picks up again at the next bar that does have a value. Returning *NaN* deliberately is therefore the way to restrict a line to the part of the chart where it means something:

```
Charts:
   _atr:         ATR(14)
   HiVolBand:    {color: red}   IF(_atr > MA(_atr, 50), C + 2 * _atr, NaN)
```

The same thing happens by itself whenever a formula cannot be evaluated on a given chart - which is why it is safe to leave trade-only items in a Charts script that is also applied to ordinary symbol charts.

A line that changes color

Each item gets one color, but two items can share one path. Define the series twice with complementary conditions, giving each the color for its own case:

```
Charts:
   _ma:     MA(C, 50)
   MaUp:    {color: green}   IF(_ma >= _ma[1] or _ma[1] >= _ma[2], _ma, NaN)
   MaDn:    {color: red}     IF(_ma < _ma[1], _ma, NaN)
```

The two items draw as one continuous moving average that is green while it is rising and red while it is falling. The extra *or _ma[1] >= _ma[2]* term lets the green segment run one bar past each turn, so the hand-off between the colors has no gap in it. Both items appear in the legend, so give them short names.

Trade Charts

When a chart is opened from a row of a **Trade List**, it carries the context of that trade. All of the *T.* elements resolve to that trade, and the **current position** elements **BarsHeld**, *FillPrice*, *FillValue* and

Shares resolve as they did during the test. That makes it possible to draw the exact stop line the strategy was using:

```
Charts:
   Trail:   {color: red, line: dashed}   Highest(H - 2 * ATR(14), BarsHeld)
```

Because *BarsHeld* counts from the entry bar of the trade being charted, *Highest(..., BarsHeld)* reproduces a trailing stop that starts fresh at entry rather than one that looks back a fixed number of bars. This example (for the long side) ships commented out in the default *charts.rts*.

To keep the line off the part of the chart before the trade began, gate it with **BarDate**:

```
   Trail:   {color: red, line: dashed}   IF(BarDate >= T.DateIn, Highest(H - 2 *
ATR(14), BarsHeld), NaN)
```

Other useful trade-chart items are the entry price as a horizontal reference (*IF(BarDate >= T.DateIn and BarDate <= T.DateOut, T.PriceIn, NaN)*) and a profit target drawn the same way.

Histogram bars and transparency

line: Bars draws the item as vertical bars rising from *base:* (0 unless you say otherwise), which is how the MACD histogram in the default script is drawn. A color given with an alpha component makes the fills partly transparent so that overlapping bars and the chart grid show through:

```
   MacdHist:   {pane: ind1, line: Bars, color: blue}   MACDH(12,26,9)
   AtrBars:    {pane: ind2, line: Bars, color: ARGB(110, 30, 90, 160)}   ATR(14)
```

The Text item

If a special Charts item called *Text* is defined, it is not plotted. It is interpreted as a **string** formula and its output is appended to the chart's legend row - the line above the price pane that shows the symbol, bar size, date and OHLC values for the bar under the crosshair.

This example adds the current bar's "volume surge" to that line:

```
Charts:
   Text:    Format("VS = {%}", V / MA(V,20) - 1)
```

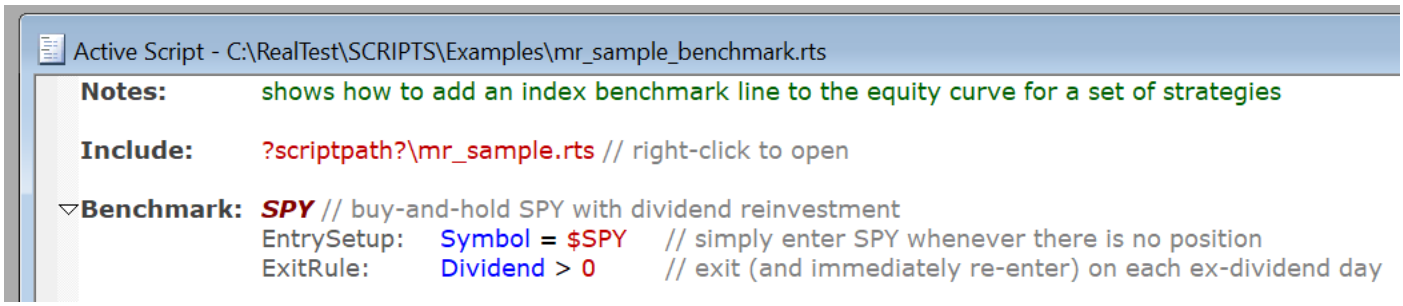
The legend then reads, for example, *AAPL - Daily 8/4/23: O = 185.52, H = 187.38, L = 181.92, C = 181.99 (-4.80%), V = 115,956,840, VS = 112.79%*.

See also **Default Scripts** and **Output Format Specification**.

14.2.13. 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:



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.

Settings can also be combined: any setting can be restated after the Include, and the last statement wins. In particular, restating a file output setting (SaveTradesAs, SaveStatsAs, ResultsFile, etc.) with a blank value clears it, so a script that includes another script can suppress an output file that the included script turns on.

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.

14.3. Formula Syntax

Most script elements accept a formula expression. These topics describe the formula language.

- **Formula Basics** - where formulas are allowed, recursion, and nan handling
- **Operators** - arithmetic, comparison, and logical operators
- **Formula Evaluation** - how and when formulas are evaluated
- **Breadth Tags / Cross-Sectional Functions** - calculations across all symbols on a date
- **Special Syntax for Individual Futures Contract Testing** - per-contract formula syntax
- **Statistics Values in Formulas** - referencing test statistics in formulas

14.3.1. Formula Basics

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.

14.3.2. Operators

The following table lists all of the operators that 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
BITNOT		2	bitwise not	
^		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
BITAND		4	bitwise AND	
BITOR		4	bitwise OR	
BITXOR		4	bitwise XOR	
LSHIFT		4	bitwise shift left	
RSHIFT		4	bitwise shift right	

+		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.

14.3.3. 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.

Short-circuit optimization in general speeds up formula processing throughout RealTest by about 50%. You might nevertheless prefer that this optimization not be employed, e.g. if you use formulas where part of the expression might result in "nan". If you need to ensure that any possible "nan" in any formula is accounted for, you can disable this optimization by unchecking *Use Short-Circuit Logic* in the **Program Options** dialog box.

14.3.4. 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 data 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.

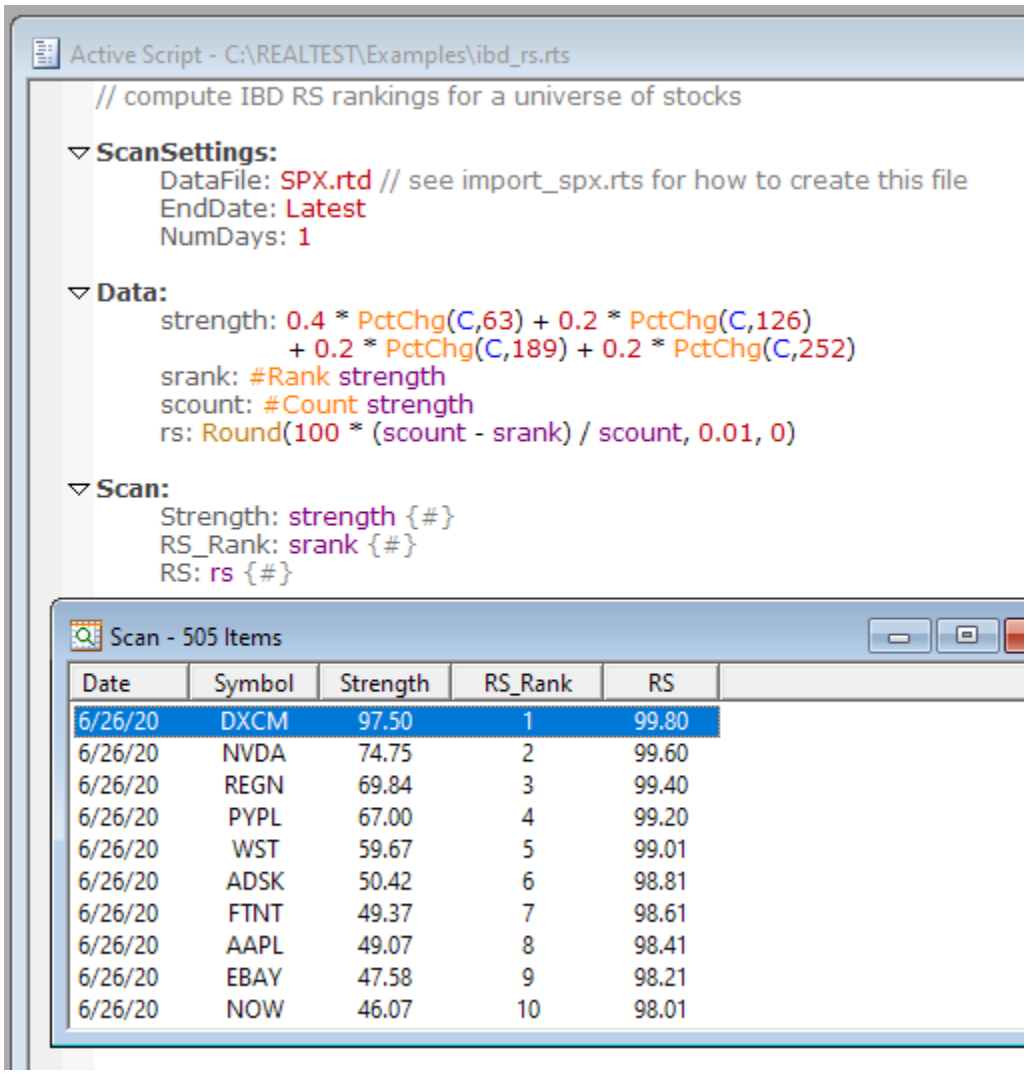
#Avg, **#Count**, **#Highest**, **#Lowest**, **#Median**, **#StdDev** and **#Sum** will, by definition, all produce the same answer for every cell in a row of data.

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:



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).

14.3.5. 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 **CSI** or **Norgate**, must have the same format used by one of them.

Specifically, the format must be either: XX-YYYYM, e.g. ES-2016Z, GC-2021K, etc. or XX_YYYYM, e.g. ES_2016Z, GC_2012K, 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 RELATIVE 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.

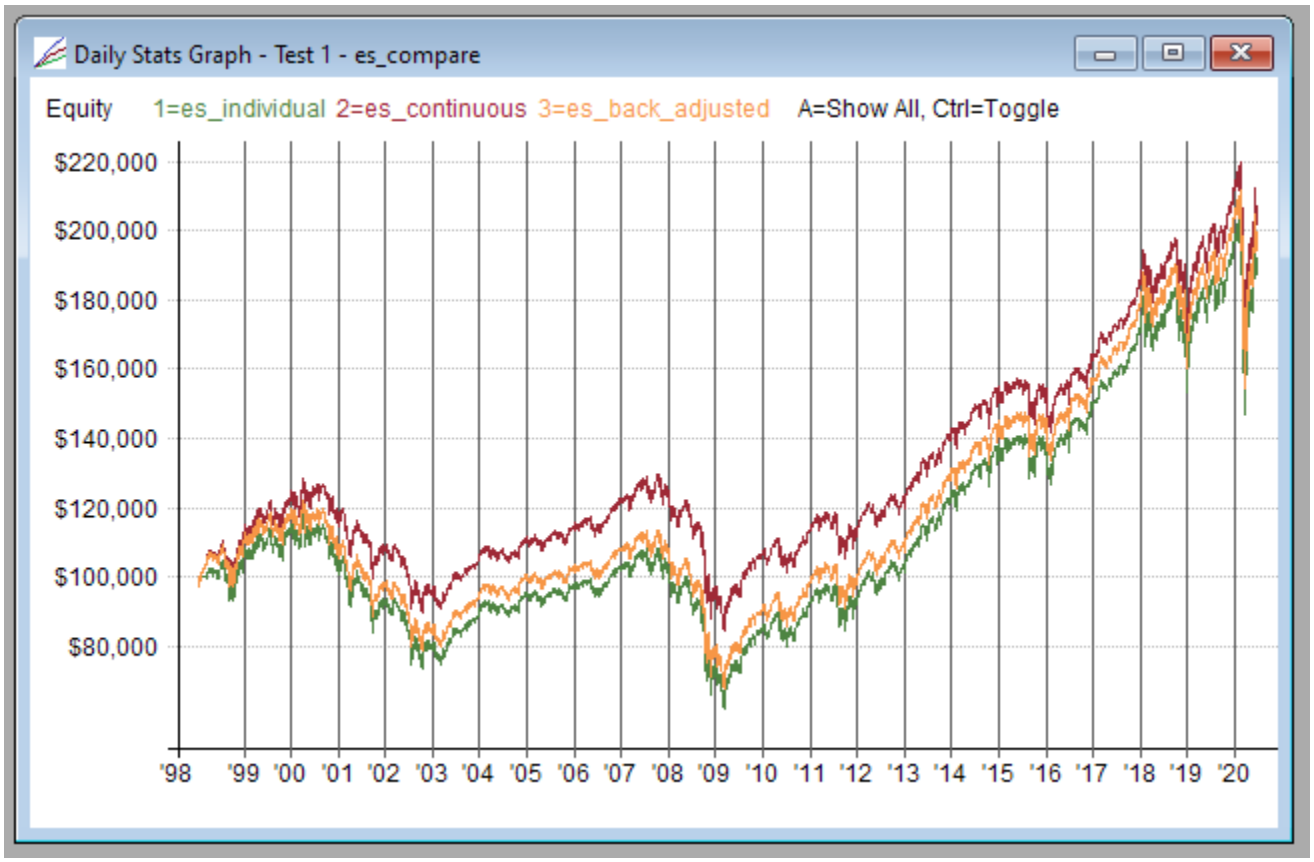
Note that a relative contract reference will return **nan** if the relevant contract does not include a bar for the current date (the date for which the formula is being evaluated).

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 **CSI** or **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.

14.3.6. 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 dollar (or other currency) value won is **S.WinAmt** and for value lost is **S.LossAmt** (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.WinAmt,S.Number) / Sum(S.LossAmt,S.Number)*. Or you might want to use a rolling 3-month profit factor, which would be *Sum(S.WinAmt,63) / Sum(S.LossAmt,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**

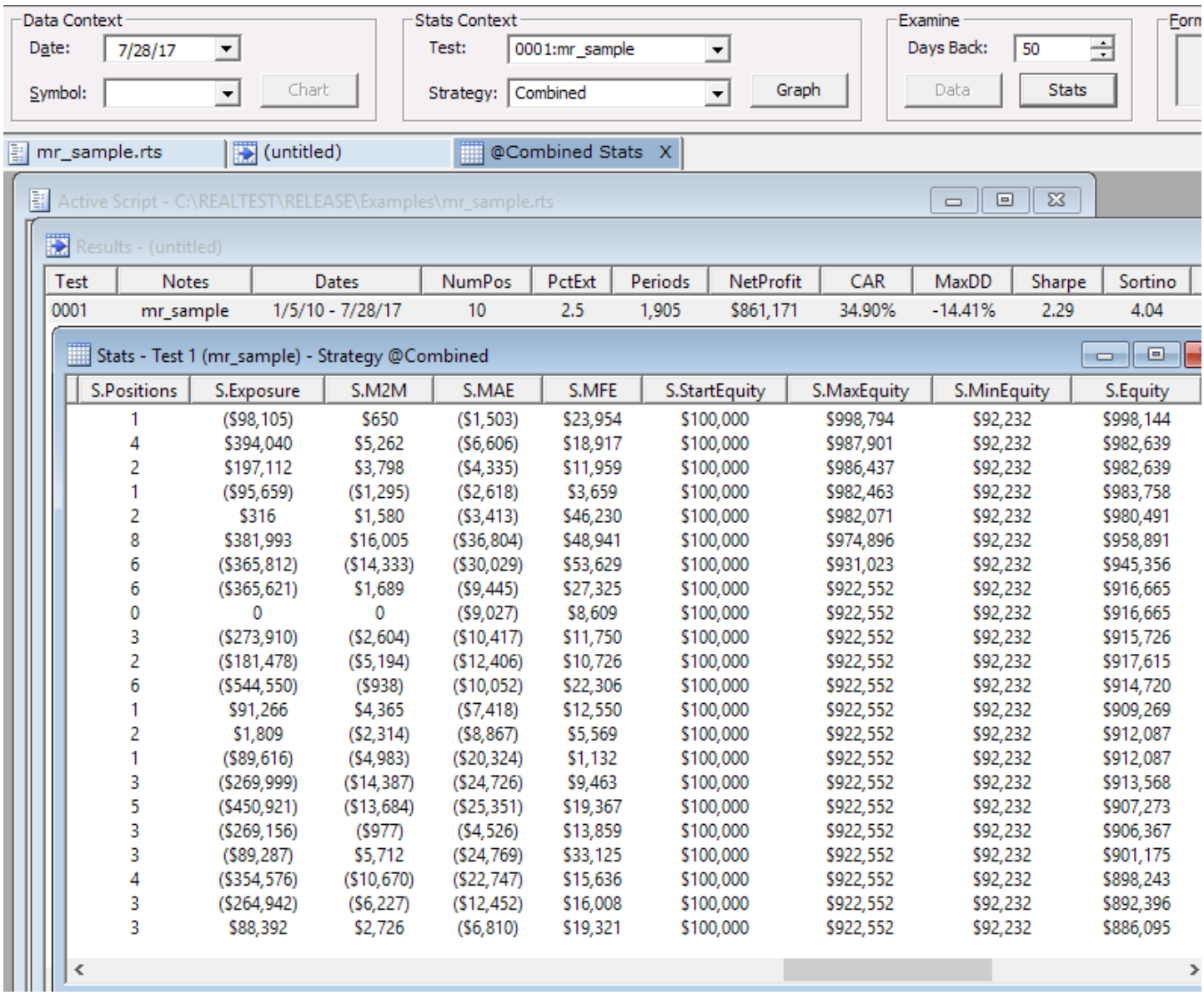
(It should be clear from the Min/Max prefixes that this is how these work.)

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.



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

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

14.4. 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.

14.5. Date Constants

Date constants in scripts or input files can be formatted in any of the following ways:

- m/d/yy (or d/m/yy if **DateInput** setting is *DMY*)
- m/d/yyyy (or d/m/yyyy likewise)
- any of the above with dashes (-) or dots (.) instead of the slashes (/)
- yyyy/mm/dd
- 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.

The **DateInput** setting determines whether x/x/y is interpreted as d/m/y vs. m/d/y.

To override the current *DateInput* setting for a **CSV import** and/or **EventListFile**, add a **CSVDateFmt** definition.

To override the current *DateInput* setting for a strategy's imported **TradeList**, add a **TLDateFmt** definition.

To specify a literal date within a formula, e.g. *EntrySetup: BarDate > 20240101 and other_conditions*, if(Date = 20200607), it must be formatted as YYYYMMDD as in this example.

To parse a date string within a formula, use the **ToDate** function.

To format a numeric date in a non-default way, use the **FormatDate** function.

To control how dates are displayed by default in the user interface or output files, use the **DateDisplay** setting.

14.6. 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.

14.7. Symbol References

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 $)
- **SymRef** - 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
- SymRef(Symbol)
- SymRef(?Symbol)
- SymRef("MSFT")

The special symbol *$CalSym* can be used to reference the **CalendarSym** of the current strategy.

See also: **Extern**.

14.8. 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-insensitive comparison is used, so for example 'A' == 'a' evaluates as true. 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**.

14.9. User-Defined Item Names

A RealTest script is essentially a collection of user-defined formulas and settings values.

Some **script sections**, such as **Import**, **Strategy**, **Settings**, and related sections contain specifically-named items that you can provide formulas or values for. These sections do not allow user-defined item names.

RealTest supports user-defined item names only in the following sections: **Charts**, **Data**, **Graphs**, **Library**, **Parameters**, **Results**, **Scan**, **StratData**, **TestData**, **TestScan**, **Trades**

Parameters is a collection of named parameter value lists. All the others are collections of named formulas.

Item names must begin with a letter or underscore and may contain only letters, digits, periods and underscores.

Item names can be no more than 80 characters in length.

Item names are used to refer to that item (to get its current value) from other formulas.

Each user-defined item name must be unique across the entire script (including those that it includes), and must not be the name of any built-in variable or function.

Within the same section type, the same name can be used for more than one item. When this occurs,

the last instance found is the only one that is kept. See **Item Name Overrides** for details about this.

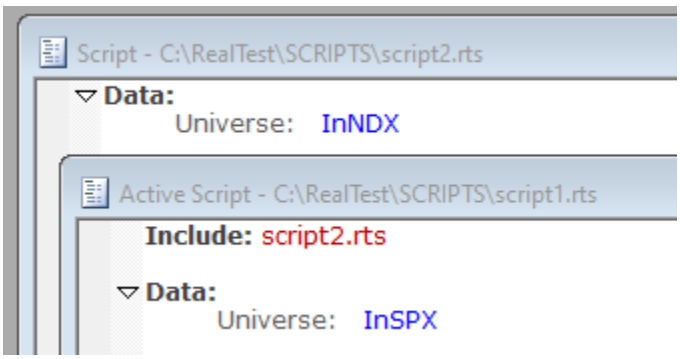
14.10. Item Name Overrides

RealTest allows user-defined item names to be re-used within the same **script section**.

RealTest does NOT allow the same name to be used in different sections, e.g. a *Data* item and a *Library* item could not have the same name.

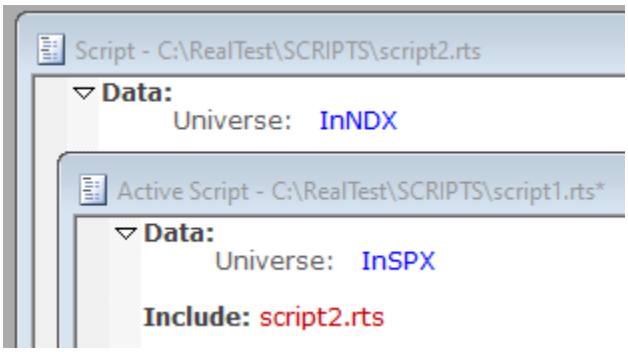
The last instance of a named formula (across all instances of the section it's defined in) becomes the formula assigned to that name.

As an example, consider the following:



The active script, *script1.rts*, will use as its universe the historical constituents of the S&P 500 index, despite including *script2.rts* which had defined *Universe* to be the Nasdaq 100 constituents.

However, if *script1.rts* is rearranged like this:



Then it will be using the Nasdaq universe.

The last encountered instance of a name becomes the definition of that name in the final combined script.

If you have some items that you don't want to be overridden, the easiest solution is to simply assign a globally-unique name to each such item.

If you prefer to use the same name for different items in different scripts without risk of override, you can use **Namespace** to differentiate them when those scripts are combined.

14.11. Script Comments

The RealTest script syntax supports three different comment formats:

- 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 to 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.

In addition to the above, RealTest also supports **Conditional Comments**. See that topic for details.

14.12. Conditional Comments

Conditional comments make it possible to designate portions of a script that are "commented out" (see **Script Comments**) in some situations and not commented out in other situations.

The syntax of conditional comments is shown in the following example:

```
#define incl_bench
#ifdef incl_bench
Benchmark: Benchmark
        Side:               Long
        Allocation:         S.Equity
        EntrySetup:         Symbol=$$XSOA.au
#endif
```

A comment condition named *incl_bench* is defined.

The *#ifdef* statement then checks to see whether *incl_bench* has been defined.

Since it has, the subsequent lines of the script are NOT commented out.

The *#endif* statement marks the end of the lines to which the prior *#ifdef* condition applies.

Compare the above to the following:

```
//#define incl_bench
#ifdef incl_bench
Benchmark: Benchmark
        Side:               Long
        Allocation:         S.Equity
        EntrySetup:         Symbol=$$XSOA.au
#endif
```

Here the line that defines *incl_bench* has been commented out explicitly. (It could also be simply deleted.)

Now *incl_bench* is NOT defined, making the *#ifdef* condition *false*, therefore the subsequent lines are not included in the script (they are commented out).

In other words, *#ifdef xxx* means "only include the subsequent lines in the script if *xxx* has been defined with *#define*".

The following lists all the syntax supported by RealTest for conditional comments:

Item	Description	Example
#define	define a conditional comment name	#define xxx
#undef	remove (un-define) a conditional comment name	#undef xxx
#ifdef	include subsequent lines if name is defined	#ifdef xxx
#ifndef	include subsequent lines if name is NOT defined	#ifndef xxx
#else	include subsequent lines if the prior #ifdef or #ifndef excluded them	#else
#endif	end the current conditional comment block	#endif

In addition to any names that you define with #define, RealTest also predefines several names that

may be useful:

Name	Description
standalone	the script is not included by another script
apply	the script is running in "Apply" mode
test	the script is running in "Test" mode
optimize	the script is running in "Optimize" mode
import	the script is running in "Import" mode
scan	the script is running in "Scan" mode
orders	the script is running in "Orders" mode
parse	the script is being validated in command-line "Parse" mode (-parse)

For example, in the screenshot above, replace *#ifdef incl_bench* with:

- *#ifdef standalone* to only include the benchmark when the script is not included by another script (which might already have its own benchmark)
- *#ifdef test* to only include the benchmark when the script is being run as a single test, not when optimizing or generating orders
- etc.

Please note that conditional comment syntax is not suggested or auto-completed by the script editor, but is (as shown above) reflected in the automatic text coloring. This shows at a glance what impact your conditional commenting has on your script.

The one case where the text coloring may not seem correct is in a script that is included by other scripts that may or may not define the name that the included script references in #ifdef or #ifndef. In this case RealTest cannot possibly know which other script you might intend to include this script with. It therefore colors the text as if any such names are not defined.

See also **Repeat Blocks** and **Combining Scripts**.

14.13. Repeat Blocks

Repeat blocks copy a group of script lines once for each value of an integer index, so a series of similar items (a filter cascade, the passes of a multi-stage smoother, a ladder of lookback periods, a family of chart lines) can be written once instead of typed out by hand. The copying is purely textual and happens before the script is parsed, so the expanded text is exactly what you would otherwise have typed.

A block starts with a *#repeat* line naming the index and its values, and ends with *#endrepeat*. Inside the block, an index name in braces such as *{k}* is a marker that is replaced by the index value in each copy. For example:

```
Data:
   out0: C
#repeat k from 1 to 4
   out{k}: AEMA(out{k-1}, 2 / 15)
#endrepeat
   final: Item("out{#}", ORDER)
```

RealTest parses this as if the script contained *out1: AEMA(out0, 2 / 15)*, *out2: AEMA(out1, 2 / 15)* and so on through *out4*. The *out0* seed item outside the block together with *out{k-1}* inside it is the usual way to chain stages without a special case for the first one, and *Item()* is the usual way to select one of the expanded items at run time when the number of stages is a **Parameter** (repeat up to the parameter's maximum value).

Syntax

The index values are given the same way as an optimization range in the **Parameters** section, but they must be integers and a range must be ascending:

Line	Description

#repeat k from 1 to 20	every integer from 1 to 20
#repeat k from 0 to 12 step 3	0, 3, 6, 9, 12
#repeat k from 5 to 80 mult 2	5, 10, 20, 40, 80
#repeat n 5, 8, 13, 21	an explicit list of values
#endrepeat	end of the block

The *def* and *round* clauses of a Parameters range are not allowed, nor are non-integer values. Both directive lines must be the first thing on their line (indentation is fine) and are removed from the expanded text.

Markers

A marker is *{expr}* with no spaces inside, where *expr* uses at least one enclosing repeat index and otherwise only integers, the operators + - * / (integer division truncates) and parentheses: *{k}*, *{k-1}*, *{2*k+1}*, *{(k-1)*3+j}*. A marker can appear anywhere in the block: in item names, formulas, Namespace names, Include paths, string literals, and comments.

Any other text in braces keeps its usual meaning, because none of it names an active index: *{#6}* or *{%}* format codes, *{color: gray}* attribute blocks, the *{x}* optimize flag, a bare *{2}*, and ordinary **comments** are all left alone. Note the flip side: a misspelled index such as *out{kk-1}* is treated as a comment, so the parser then sees *out* and reports whatever that produces. A marker that names an index but cannot be evaluated (*{k/0}*) is an error.

Parameter names are not allowed in a marker. The block is expanded once when the script is parsed, while a parameter takes a different value in every optimization test. Repeat up to the parameter's maximum and select at run time with *Item()*, as in the first example.

Nesting

Blocks can be nested. Each level needs a different index name, and since the inner *#repeat* line is substituted like any other line, an inner range can depend on the outer index:

```
Library:
#repeat k from 1 to 4
#repeat j from 1 to {k}
   w{k}_{j}: {k*10+j}
#endrepeat
#endrepeat
```

Blocks that span sections

A block may contain section headers, so instead of putting the index in every item name you can put it in a **Namespace** and keep the names inside plain (restate the section header after the Namespace line):

```
Namespace: pass0
Data:
   out: C
#repeat k from 1 to 4
Namespace: pass{k}
Data:
   a: AEMA(pass{k-1}.out, 2 / 15)
   b: AEMA(a, 2 / 15)
   out: a + 0.5 * (a - b)
#endrepeat
Namespace: none
Data:
   final: Item("pass{#}.out", ORDER)
```

The usual Namespace rules apply: inside a namespace only that namespace's own items are visible by bare name, so Data, Library and Parameter items defined outside it must be written inline or referred to by a dotted name such as *pass2.out*, and a block that sets a namespace leaves it set after *#endrepeat*, exactly as pasting the text would.

An **Include** line inside a block is repeated per iteration with its path (and any preceding Namespace line) substituted, but the included file is parsed on its own and does not see the enclosing index.

Errors and the editor

A syntax error inside a block is reported at the line of the script as written, with the failing copy of the block appended to the message, for example *(#repeat k = 3)*. The Data window and the script copy

saved with a test show the expanded item names (*out1*, *out2*, ...), which is usually what you want when checking a cascade.

The script editor colors a name containing a marker (*p{k}a*, *out{k-1}*) as the item it will become, and auto-complete offers the expanded names. A Notes line that begins with the word *#repeat* is treated as the directive, the same rule that applies to *#ifdef*.

See also **Conditional Comments**, **Namespace** and **Combining Scripts**.

14.14. 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.

Either slash direction may be used as the path separator. Forward slashes (/) and backslashes (\) are interchangeable everywhere RealTest accepts a path -- in the script parser, in path-expansion variables such as ?scriptpath?/file.csv, in editor path auto-complete, and in the right-click "open file under cursor" command. A drive-letter path such as c:/RealTest/Data is recognized the same as c:\RealTest\Data.

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 **OrderClerkFolder** setting
?scriptpath?	the full path of the current script*, without the name
?scriptname?	the name of the current script*, without the full path or the .rts extension
?mainscriptpath?	the full path of the top-level script*, without the name
?mainscriptname?	the name of the top-level script*, without the full path or the .rts extension
?testname?	the name of the test just run (for SaveStatsAs etc.)
?desktop?	the full path to your Windows desktop folder
?realtest?	the full path of the folder where RealTest is installed
?data?	the full path of the Data Folder specified in View / Program Options
?output?	the full path of the Output Folder specified in View / Program Options
?scripts?	the full path of the Scripts Folder specified in View / Program Options

Additionally, any Windows environment variable can be specified, e.g. ?programdata?, ?onedrive?, etc. Like the items in the table above, these expand wherever they appear in the path - as the leading

component, as a folder in the middle, or within the file name itself.

If a ?token? matches neither an item in the table nor an environment variable, the script will not run: RealTest reports "unrecognized ?token? in path" on that line. A question mark cannot appear in a Windows file name, so a path still holding one could never be created - reporting it is better than silently producing no output.

* "top-level script" means the script being run (which might include other scripts) and "current script" means the specific script that the expansion variable was typed in. These will typically be the same script unless **Include** is involved. If script A includes script B (from a different folder), and script B refers to ?scriptpath?, it will expand to script B's path. If script B refers to ?mainscriptpath?, it expands to script A's path.

14.15. 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.

Two forms are supported within the *{ }* block: the short-glyph form (e.g. *{$0}*) documented in the first table below, and the newer named-attribute form (e.g. *{format: dollar, places: 0}*) documented in the second. The two forms cannot be mixed in the same comment block, but the same formula may contain multiple comment blocks.

Short-Glyph Form

Code	Used In	Shows This	Digits (if present) Specify
#	scans, trades, charts, graphs, results	general-purpose numeric value	number of decimal places
#x	scans, trades, charts, graphs, results	integer in hexadecimal format	n/a
##	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 in 32nds, as futures prices are quoted (e.g. 110'16)	n/a
~	scans, trades, charts, graphs, results	value as *True* or *False*	n/a
^	charts	show this indicator in an upper indicator pane (^ = *^1* = pane closest to price; *^2* and *^3* select additional panes stacked above)	optional pane number 1, 2 or 3

^	graphs	include this graph in Test Summary Reports	n/a
|	charts	show this indicator as a line in the volume pane (use *line: Bars* to draw actual bars)	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.

Named-Attribute Form

An output format comment may alternatively contain a comma-separated list of *name: value* attributes, e.g. *{format: dollar, places: 0, color: orange}*. The script editor pops a context-aware autocomplete list when the cursor is inside the *{ }* block, showing only the attributes that apply in the current script section.

Attribute	**Alias**	**Used In**	**Value**
format	*unit*	scans, trades, charts, graphs, results	one of: *number*, *plain*, *dollar*, *percent*, *fraction*, *date*, *bool*, *hex*
places	*decimals*	scans, trades, charts, graphs, results	integer number of decimal places
negred	–	scans, trades, charts, graphs, results	*1* or *0* – show negative values in red
name	*label*	scans, trades, charts, graphs, results	quoted display label that overrides the item name
color	–	charts (per indicator); strategy / template / benchmark / statsgroup / combined section headers	named color, *#RRGGBB*, *RGB(r,g,b)*, *#RRGGBBAA*, or *ARGB(a,r,g,b)* – overrides the default palette color; the alpha forms make bar fills partially transparent (see below)
pane	–	charts	one of: *price*, *ind1*, *ind2*, *ind3*, *volume* – selects which chart pane to draw the indicator in
line	*linetype*	charts, graphs	one of: *Line*, *Bars*, *Dashed*, *Dotted* (graphs treat *Dashed/Dotted* as *Line*)
width	–	charts	integer *1* to *20* – pen width in pixels for this line, overriding the Line Width chart option (with *line: Bars*, sets each bar's half-width)
base	–	charts, graphs (with *line: Bars*)	baseline (x-axis) value for histogram-style rendering
hat	–	charts, graphs	*1* or *0* – equivalent to the legacy ^ glyph (route to indicator pane in charts; include in Test Summary Reports in graphs)
factor	–	trades	*1* or *0* – marks the item as a **Factor Scan** factor and evaluates it as of the trade's setup bar rather than its exit bar

The *1/0* attributes (*negred*, *hat*, *factor*) may also be given as a bare word meaning 1: *{factor}* is the same as *{factor: 1}*, and *{factor, format: percent, places: 2}* combines it with other attributes.

Color Values

The *color:* attribute accepts five value forms:

1. a named color: *black, white, red, darkred, green, darkgreen, blue, darkblue, cyan, magenta, yellow, orange, purple, brown, gray, darkgray, lightgray*
2. an HTML-style hex code: *#RRGGBB* (e.g. *#FF8800*)
3. an RGB triple: *RGB(r, g, b)* with each component 0–255
4. a hex code with an alpha (opacity) byte: *#RRGGBBAA* (e.g. *#FF880080* for half-transparent orange)
5. an ARGB quad: *ARGB(a, r, g, b)* with the alpha (opacity) component first

The alpha component is an opacity from 1 (nearly invisible) to 255 (fully opaque); omitting it, or giving 0 or 255, draws the color fully opaque. Alpha affects only **fills**: chart indicators drawn with *line: bars* and the bars of bar-style graphs. Lines and text always draw opaque. Partially transparent fills let overlapping bars and the chart grid show through, which is especially useful for multi-strategy drawdown graphs. Bar graphs can also be given a default transparency without editing any color specs – see the Bar Graph Opacity setting in the **graph options dialog**.

On the **Strategy**, Template, Benchmark, StatsGroup, and Combined section header lines, a *{color: ...}* comment overrides the color used for that strategy in graphs, result columns, trade overlays on charts, and combined-strategy reports.

14.16. New Script Template

Selecting *New Script* from the **File Menu** or clicking the corresponding **Tool Bar** button creates a new blank **Script Window**.

If you'd prefer your new scripts to have some preliminary content rather than being blank:

- create a new script with your desired default content

- save this script as **newscript.rts** in your RealTest program folder (c:\RealTest by default)

Now, whenever you create a new script window, it will begin with that template.

14.17. Combining Scripts

A RealTest script is not really describable as "code", or "a program". Rather, it is a collection of named values and formulas that serve as the input parameters to a general-purpose backtesting engine. These items are organized in **Script Sections**.

Script sections can be combined in a single script and/or kept in separate scripts that are combined using **Include**.

Either way, there is logically one big script, so we'll refer to this as "the script" or "the combined script".

It mostly does not matter in what sequence the various sections appear within the combined script.

The sequence of items within a section matters for some section types and not for others.

Each script section has its own special purpose and content, with slightly different rules regarding combination and sequencing.

The categories of script sections are:

Category	Sections	Items	Override	Script Sequence	Item Sequence

Data Import	**Import**	have specific names and purposes	mixed	doesn't matter	DataSource must come first, last instance wins for overrides, else doesn't matter
Data Calculation	**Data**, **TestData**, **StratData**	have user-defined names and formulas	optional	doesn't matter	is calculation sequence
Data Output	**Results**, **Graphs**, **Trades**, **Charts**, **Scan**, **TestScan**	have user-defined names and formulas	optional	doesn't matter	is display sequence
Formula Library	**Library**	has user-defined names and formulas	optional	doesn't matter	doesn't matter
Parameter Values	**Parameters**	has user-defined names and value lists	always	doesn't matter	is optimization sequence
Strategy Definition	**Strategy**, **BenchMark**, **StatsGroup**, **Combined**, **Template**	have specific names and purposes	never	determines default daily processing sequence in tests	doesn't matter
Script Settings	**Settings etc.**	have specific names and purposes	always	doesn't matter	last instance wins for overrides, else doesn't matter

Each of the above sections may appear zero or more times in the combined script. (Some **Run Modes** are only available when the script defines the section to which the mode pertains, e.g. *Import* for import mode, *Strategy* for test mode, etc.)

The sections that have specific names and purposes (*Import, Strategy, Settings*) do not allow user-defined items to be added. The script editor will show you which specific items are available as you are typing. Highlight any item name after typing it and then press F1 to learn more about it.

In some cases, specific items are formulas (e.g. **EntrySetup** in a strategy), in other cases they are some type of constant setting value (e.g. **EntryTime** in a strategy -- values from a finite list of choices), and in some cases they are folder or file paths (e.g. **SaveAs** in import).

The sections that have user-defined names all have the same format, which is*name: definition.* Name is any name of your choosing, provided it was not already used in a different section with user-defined names and is not a built-in variable name (reserved word) such as "close".

For sections that allow overrides, writing a new item with the same name as a prior one replaces the prior item definition with the new value or formula. This might be done explicitly in the same script or implicitly by including a script that redefines the item.

The *Import* section is a special case because some of its items logically can have multiple instances, such as *DataSource*, *DataPath*, *IncludeList*, etc. while other items must be defined once for the entire import, such as *StartDate*, *EndDate*, *SaveAs*, *Padding*, *KeepAdjusted*. Combining Import sections from different scripts can easily lead to complexity and must therefore be given careful design consideration and testing.

All the other sections that have items with specific names always allow override of any item by defining it again with a different value. This facilitates combining scripts that define the same items, especially settings.

The sections that allow optional name override disallow it by default -- the parser will show an error message if a name is reused and the script doesn't specify **AllowSameName: True** in *Settings*.

If this setting is specified then items in any of those sections can be replaced by defining a different item with the same name in the same section type. (It is always an error to try to use the same name in different section types.)

For special use cases where you would like to use the same name in various scripts, sometimes combine those scripts, and NOT override any user-defined names, RealTest provides an optional **Namespace** mechanism. See that topic for details.

Conditional Comments can also be very helpful when combining scripts.

14.18. Default Scripts

RealTest uses four default scripts:

- **Results.rts** - provides **Results Section** formulas defining the columns to display in **Results Windows**
- **Graphs.rts** - provides **Graphs Section** formulas defining the types of graphs available in **Graph Windows**
- **Trades.rts** - provides **Trades Section** formulas defining the columns to display in **Trades Windows**
- **Charts.rts** - provides **Charts Section** formulas defining the indicators to be added to **Chart Windows**

These four scripts are automatically included in any script that is parsed IF that script (or other scripts it includes) does not already define the corresponding section. For example, if your script has no Results section (as it typically will not), then the default Results.rts is automatically included. This ensures that the contents of the four window types listed above are always defined.

When a default script needs to be included, RealTest looks for it in the following locations:

1. The folder of the script being parsed
2. The folder in which RealTest is installed (c:\RealTest by default)

The RealTest Installer puts the "factory default" versions of these four scripts in your installation folder to ensure that they're always available. It is often a good practice to copy these to your Scripts folder before making changes, so that your "working" versions will be with your own scripts.

When you define any of the above four sections in any of your own scripts, this means that the corresponding default script will NOT be included.

For example if you add a Results section to a script with some strategy definitions, the columns defined by that section are the only columns that will appear in the Results window.

If you would like to include the default items AND add a few of your own, you can do so as in this example:

```
StratData:
        // tracker benchmark rank factor
        factor:         S.Equity / MA(S.Equity, 50)
        // tracker benchmark rank number
        srank:          #Rank  if(IsTracker, factor, nan) // nan excludes strategies themselves

Include: c:\RealTest\Graphs.rts

Graphs:
        StratRank: srank
```

This would add a graph of your auto-tracker benchmark rank numbers to the end of your set of stats graphs.

If you prefer your script-specific items to appear before the default ones, simply move the Include line to after the Graphs section.

It is most often necessary to add custom items to the default section definitions when you want to refer to variables defined by your script, such as the "srank" StratData item in the above example (from oex_tf_auto_track.rts).

See each of the links above for more information about the specifics of each default script section and its corresponding output window type.

14.19. Syntax Element Categories

All of the elements of the RealTest Script Language syntax are listed here according to usage categories, alphabetically within each category.

14.19.1. 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
- **Combined** - all-strategy combined constraints such as MaxExposure, MaxInvested, and MaxPositions
- **Data** - named formulas calculated and stored in memory arrays for each bar of each symbol 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
- **Namespace** - specifies a prefix for script-defined names, to prevent duplication when including multiple scripts
- **Notes** - a free-form section in which to organize any notes about the script
- **OptimizeSettings** - settings that apply only when the script is run as Optimize
- **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
- **StatsGroup** - defines a group of strategies to combine for test statistics and multi-strategy constraints
- **StratData** - named formulas calculated and stored in memory arrays for each strategy each date while tests are run
- **Strategy** - begin a new trading strategy definition
- **Template** - begin a new template strategy definition
- **TestData** - named formulas calculated and stored in memory arrays for each symbol each date while tests are run
- **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

14.19.2. Settings

These define the settings to apply when running a script.

- **AccountSize** - starting capital amount
- **AccountType** - specifies the type of account being modeled
- **AllowSameName** - allow user-defined items to replace prior ones with the same name in the same section type
- **AlwaysImport** - causes data to be re-imported before a script runs in any mode
- **BarSize** - default test data timeframe
- **CashIntPct** - interest rate received for positive daily excess cash
- **ChartBarSize** - bar size charts display
- **ChartCandlestick** - draw candlesticks rather than OHLC bars
- **ChartEntryExitLine** - draw a line connecting entry and exit on trade charts
- **ChartGrid** - show grid lines in charts
- **ChartImageHeight** - pixel height of chart images saved by SaveScanChartsTo and SaveTradeChartsTo
- **ChartImageWidth** - pixel width of chart images saved by SaveScanChartsTo and SaveTradeChartsTo
- **ChartIndicatorPane** - show the upper indicator pane(s) in charts
- **ChartLineWidth** - chart indicator line width in pixels
- **ChartLogScale** - use a logarithmic price scale in charts

- **ChartShowOtherTrades** - also show the symbol's other trades on a trade chart
- **ChartSplitAdjust** - show split-adjusted prices in charts
- **ChartTradeExtra** - extra bars a trade chart shows beyond the trade
- **ChartTradeExtraAfter** - whether a trade chart ends ChartTradeExtra bars after the trade's Entry or Exit
- **ChartTradeLabels** - which trade markers get text labels on a trade chart: None, Selected or All
- **ChartVisibleBars** - minimum number of bars visible in a chart
- **ChartVolumePane** - show the volume pane in charts
- **CommCurrency** - how the Commission formula is interpreted in a multi-currency script
- **Currency** - account base currency for multi-currency system models
- **DataFile** - test data file path
- **DateDisplay** - how dates are displayed in the user interface, scans, trade lists, etc.
- **DateInput** - how dates are parsed when day/month vs. month/day is ambiguous
- **DateSym** - symbol to use for "Earliest" or "Latest" in StartDate or EndDate
- **DaysPerYear** - number of market days per year for stats purposes
- **DividendsToCash** - count dividends accrued to open positions as free cash from the ex-date
- **EndDate** - last date of test
- **EndOfTestExits** - controls whether open positions are exited at the end of a backtest
- **ExchangeMap** - details about specific exchanges for which you may want to generate orders
- **HolidayList** - list of holidays (only required for order generation)
- **KeepTrades** - types of trades to store in each results record
- **LogDetail** - which categories of output to include in the Test Details Log
- **MarginIntPct** - interest rate charged for negative daily excess cash (margin loan)
- **NumBars** - number of market dates to test
- **OptAnchor** - whether and how to anchor the start or end date of multi-interval optimizations / walk-forward
- **OptByStrategy** - run the optimization separately for each strategy
- **OptBySymbol** - run the optimization separately for each symbol
- **OptClearResults** - clear the active results window before running an optimization
- **OptContinueResults** - resume a stopped optimization from the tests already in ResultsFile (Combinatorial mode only)
- **OptimizeMode** - the type of optimization to perform (Combinatorial, Sequential, Genetic, Random, Defaults Only)
- **OptIterations** - run the configured test multiple times (required by Genetic and Random modes)
- **OptKeepBest** - keep only the N best-scoring tests (requires OptSortResults)
- **OptKeepByDate** - apply OptKeepBest to each date interval separately
- **OptKeepBySymbol** - apply OptKeepBest to each For Each Symbol test
- **OptKeepNoTrades** - keep result rows for parameter combinations that produced zero trades
- **OptMultiSaveAs** - perform SaveStatsAs and/or SaveTradesAs once per test in the optimization
- **OptMutatePct** - mutation probability for Genetic optimization

- **OptNoDialog** - run the optimization without displaying the Optimize dialog first
- **OptNoStats** - drop per-period stat details from each test to allow much larger optimization runs
- **OptOverfit** - run an Overfitting Analysis (CSCV/PBO) when the optimization completes
- **OptOverfitBlocks** - number of time blocks for the Overfitting Analysis (even, 4 to 20)
- **OptScoreCol** - name of the Results column to use as the optimization score (fitness function)
- **OptShards** - launch N child processes to parallelize a Combinatorial optimization across CPU cores
- **OptShowResults** - when to display the results window during a run (Live, AtEnd, None)
- **OptSortResults** - re-sort the result rows by OptScoreCol after each test
- **OptSummaryLog** - display a summary log when the optimization run finishes
- **OptTestInterval** - spacing between date intervals (in OptTimeUnit units)
- **OptTestLength** - length of each date interval (in OptTimeUnit units)
- **OptTimeUnit** - time unit for multi-interval optimizations or walk-forward (None, Years, Months, Weeks, Days)
- **OptWalkForward** - generate a walk-forward test from the multi-interval optimization
- **OrderClerkFolder** - path of folder to use with OrderClerk
- **OrdersComment** - string to add to the top of the order list when orders are generated
- **OrdersFile** - path\name of order list file to generate
- **OrdersFreeCash** - path\name of a text file containing the current free cash value of a brokerage account
- **OrdersLiveData** - allows order generation for "ThisClose" entry and exit times
- **OrdersMktAsLmtPct** - allows generated market orders to optionally be converted to limit orders
- **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
- **OrdersUseQtyIn** - forces generated exit order quantities to match the position's entry quantity
- **RandomSeed** - provides a way to use the same sequence of random numbers every time a script is run
- **ReportContent** - specify which content items to include in summary reports
- **ReportsFolder** - path of folder in which Test Summary Reports are create
- **ResultsByYear** - add per-year results columns for the listed stats (ROR, MaxDD, Trades)
- **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
- **SaveLogAs** - path\name of a text file to create and write all log output produced during the run to
- **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
- **SaveScanChartsMax** - maximum number of charts for SaveScanChartsTo to save
- **SaveScanChartsTo** - path of folder in which to automatically save a chart for every row of a scan or TestScan list

- **SaveStatsAs** - path\name of CSV file to create and write stats details to
- **SaveStatsInclude** - which strategy types to include in the SaveStatsAs CSV output file
- **SaveTestListAs** - path\name of CSV file to create with the list of test results as they appear in the results window
- **SaveTradeChartsMax** - maximum number of charts for SaveTradeChartsTo to save
- **SaveTradeChartsTo** - path of folder in which to automatically save a chart for every trade of a test or orders run
- **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
- **ShortLoanIntPct** - annualized interest rate paid on short positions (short borrow fee rate)
- **ShortPrcdsIntPct** - interest rate received on proceeds from short positions
- **SkipTestIf** - allows tests to be skipped in multi-parameter optimizations (e.g. useless parameter combinations)
- **StartDate** - first date of test
- **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 ouput rows for every date of a test, not just the last date
- **TestScanPositions** - runs the TestScan for all open positions rather than for all symbols
- **UseAvailableBars** - allows simple averages and indicators to optionally be calculated with fewer bars than specified

14.19.3. Optimize Settings

These settings configure **Optimize mode** (multiple back-to-back tests). They are specified in an **OptimizeSettings** section and correspond to the controls in the Optimize dialog. Optimize settings specified in a script are shown in the dialog when it opens, and the dialog's **Save to Script** button writes the dialog's current settings back into the section for you.

- **OptAnchor** - whether and how to anchor the start or end date of multi-interval optimizations / walk-forward
- **OptByStrategy** - run the optimization separately for each strategy
- **OptBySymbol** - run the optimization separately for each symbol
- **OptClearResults** - clear the active results window before running an optimization
- **OptContinueResults** - resume a stopped optimization from the tests already in ResultsFile (Combinatorial mode only)
- **OptimizeMode** - the type of optimization to perform (Combinatorial, Sequential, Genetic, Random, Defaults Only)

- **OptIterations** - run the configured test multiple times (required by Genetic and Random modes)
- **OptKeepBest** - keep only the N best-scoring tests (requires OptSortResults)
- **OptKeepByDate** - apply OptKeepBest to each date interval separately
- **OptKeepBySymbol** - apply OptKeepBest to each For Each Symbol test
- **OptKeepNoTrades** - keep result rows for parameter combinations that produced zero trades
- **OptMultiSaveAs** - perform SaveStatsAs and/or SaveTradesAs once per test in the optimization
- **OptMutatePct** - mutation probability for Genetic optimization
- **OptNoDialog** - run the optimization without displaying the Optimize dialog first
- **OptNoStats** - drop per-period stat details from each test to allow much larger optimization runs
- **OptOverfit** - run an Overfitting Analysis (CSCV/PBO) when the optimization completes
- **OptOverfitBlocks** - number of time blocks for the Overfitting Analysis (even, 4 to 20)
- **OptScoreCol** - name of the Results column to use as the optimization score (fitness function)
- **OptShards** - launch N child processes to parallelize a Combinatorial optimization across CPU cores
- **OptShowResults** - when to display the results window during a run (Live, AtEnd, None)
- **OptSortResults** - re-sort the result rows by OptScoreCol after each test
- **OptSummaryLog** - display a summary log when the optimization run finishes
- **OptTestInterval** - spacing between date intervals (in OptTimeUnit units)
- **OptTestLength** - length of each date interval (in OptTimeUnit units)
- **OptTimeUnit** - time unit for multi-interval optimizations or walk-forward (None, Years, Months, Weeks, Days)
- **OptWalkForward** - generate a walk-forward test from the multi-interval optimization

14.19.4. Scan Settings

These settings take effect only when a script is run in **Scan** mode; they are ignored in other run modes. Specify them either in the general **Settings** section or a **ScanSettings** section.

- **SaveScanChartsTo** - path of folder in which to automatically save a chart for every row of the scan (in a test or orders run, of the TestScan list)
- **SaveScanChartsMax** - maximum number of charts for SaveScanChartsTo to save
- **SaveScanAs** - path\name of CSV file to create and write scan output to
- **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

14.19.5. Order Settings

These settings take effect only when **generating orders**; they are ignored in other run modes. Specify them either in the general **Settings** section or an **OrderSettings** section.

- **ExchangeMap** - details about specific exchanges for which you may want to generate orders

- **HolidayList** - list of holidays (only required for order generation)
- **OrderClerkFolder** - path of folder to use with OrderClerk
- **OrdersClassDelim** - how a symbol class suffix (e.g. BRK.B) is written in generated order symbols
- **OrdersComment** - string to add to the top of the order list when orders are generated
- **OrdersFile** - path\name of order list file to generate
- **OrdersFreeCash** - path\name of a text file containing the current free cash value of a brokerage account
- **OrdersLiveData** - allows order generation for "ThisClose" entry and exit times
- **OrdersMktAsLmtPct** - allows generated market orders to optionally be converted to limit orders
- **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
- **OrdersUseQtyIn** - forces generated exit order quantities to match the position's entry quantity

14.19.6. 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
- **KeepNonIncluded** - keep all symbols found in local file data path
- **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
- **PadAlignSym** - symbol to use for Padding: AlignWithSymbol
- **Padding** - type of padding (if any) to use for missing bars
- **RemoveLatest** - removes global latest bar from any symbol that has it, or only in-progress session bars (Partial)
- **RepairData** - whether to detect and repair common data errors during import
- **SaveAs** - path\name of imported data (.RTD) file to save
- **StartDate** - earliest start date
- **SymInfoFile** - symbol information file path
- **Update** - Norgate data update request

14.19.7. Strategy Elements

All the elements of a trading strategy definition.

- **Allocation** - dollars allocated to this strategy
- **AllowMissingBar** - allow backtests to trade and orders to be generated after missing bars
- **AllowNoVolume** - allow entry and exit transactions on zero-volume days (typically padding bars)
- **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 (excluded from backtest stats)
- **CashList** - CSV file with list of specific deposit and/or withdrawal amounts (excluded from backtest stats)
- **Category** - position category value for use with **MaxCatExp**, **MaxCatInv**, and **MaxSameCat**
- **CloseSlip** - slippage for at-close market transactions
- **Collateral** - dollar collateral (margin) required to open a futures, FX, or crypto position (alias: FuturesMargin)
- **Commission** - commission per entry or exit transaction in dollars
- **Compounded** - optionally overrides the automatically-determined **S.Compounded** setting
- **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
- **DynamicSizing** - determines whether strategy operates in a special dynamic sizing mode
- **EntryLimit** - price to use when entering a position with a limit order
- **EntryPhase** - setup processing phase (lower phases finalize their entry orders first)
- **EntryScore** - ranking value for entry-time constraints (does not impact setup selection)
- **EntrySetup** - true/false condition for whether a stock is a setup this period

- **EntrySkip** - condition for whether to skip an entry (does not impact setup selection)
- **EntryStop** - price to use when entering a position with a stop order
- **EntryTime** - logical time of day at which positions are to be entered (typically next open)
- **EntryTradeValue** - calculates a value to store in **T.ValueIn** item in the trade list record for this entry
- **ExitLimit** - price to use when exiting a position with a limit order
- **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** - true/false condition for whether to exit a position
- **ExitScore** - score for ranking open positions prior to evaluating exit criteria each day
- **ExitStop** - price to use when exiting a position with a stop order
- **ExitStopQty** - share or contract quantity for partial stop-price exits
- **ExitStopTime** - exit stop execution time
- **ExitTime** - logical time of day at which positions are to be conditionally exited (typically next open)
- **ExitTradeValue** - calculates a value to store in **T.ValueOut** item in the trade list record for this exit
- **FeesInOut** - daily deposit and/or withdrawal specification by formula (included in backtest stats)
- **FeesList** - CSV file with list of specific deposit and/or withdrawal amounts (included in backtest stats)
- **GreedyScoring** - determines whether strategy uses a special variant of the setup selection process
- **HolidayList** - list of holidays (only required for order generation)
- **IgnoreDividends** - exclude dividends from trade and test P/L calculations
- **LimitExtra** - extra excursion beyond a limit price required to assume order was filled
- **LimitSlip** - slippage for at-limit-price transactions (typically 0)
- **MarkToMarket** - whether strategy equity includes open-position mark-to-market value
- **MaxCatExp** - maximum same-category percent investment
- **MaxCatInv** - maximum same-category dollar investment
- **MaxEntries** - maximum position entries to allow (rarely used)
- **MaxExposure** - maximum percent investment
- **MaxInvested** - maximum dollar investment
- **MaxLongExp** - maximum long percent investment
- **MaxLongInv** - maximum long dollar investment
- **MaxNetExp** - maximum net long-short percent investment
- **MaxNetInv** - maximum net long-short dollar investment
- **MaxNewExp** - maximum new exposure per day
- **MaxNewInv** - maximum new investment per day
- **MaxNewPos** - maximum new positions per day

- **MaxPerTurn** - how many setups per selection turn a strategy can add
- **MaxPositions** - maximum open positions
- **MaxSameCat** - maximum same-category position count
- **MaxSameSym** - maximum same-symbol position count
- **MaxSetups** - maximum entry setups per day
- **MaxShortExp** - maximum short percent investment
- **MaxShortInv** - maximum short dollar investment
- **MaxSymExp** - maximum same symbol percent investment
- **MaxSymInv** - maximum same-symbol dollar investment
- **MinFreeCash** - minimum free cash in the account
- **MinNetExp** - minimum net long-short percent investment
- **MinNetInv** - minimum net long-short dollar investment
- **OpenSlip** - slippage for at-open market transactions
- **OrdersMktAsLmtPct** - 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 a strategy-specific order list file to generate
- **OrdersFreeCash** - path\name of a text file containing the current free cash value of a brokerage account
- **OrdersIgnoreSplits** - treat split factors as dividend adjustments and ignore them when generating orders
- **OrdersNetLiq** - path\name of a text file containing the current live Net Liquidation Value of a brokerage account
- **OrdersTemplate** - path/name a strategy-specific orders template file
- **OrdersUseTLEndDate** - whether TLEndDate is applied in Orders mode for this strategy
- **PriceRound** - order and trade price rounding interval
- **QtyFinal** - can be used to modify the quantity of an entry after setup ranking is complete
- **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 to enter as shares, value, or percent depending on QtyType
- **Reduce** - whether to reduce position size rather than skip the entry due to capacity constraints
- **RollCost** - total cost of each roll transaction while holding a futures position
- **SetupScore** - ranking factor value used to prioritize entry setup selection
- **SetupSkip** - entry setup skip condition
- **Side** - strategy side constant
- **Slippage** - general-purpose slippage (price points)
- **StartPercent** - percent of initial AccountSize to allocate to this strategy
- **StopSlip** - slippage for at-stop-price transactions
- **StrategyScore** - ranking factor value for strategies at each turn of the setup selection process

- **TLAdjusted** - whether quantities and prices in an imported trade list are split-adjusted
- **TLAdoptPos** - whether trades entered before the test's StartDate are adopted by the strategy
- **TLCalcZPrices** - whether theoretical zero-slippage prices are calculated for Hybrid Trade List strategies run in Orders mode
- **TLDateFmt** - whether dates in an imported trade list are DMY or MDY
- **TLDelim** - specifies the column delimiter in a tradelist file
- **TLFields** - defines the column layout of an imported trade list CSV file
- **TLEndDate** - date to switch from trade list playback to strategy formula processing in Test mode
- **TLNumFmt** - specifies the number format in a tradelist file
- **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
- **Tracker** - whether to automatically create a tracking benchmark copy of a strategy
- **TradeList** - imported trade list file (CSV format)
- **Using** - strategy/benchmark/template to inherit from

14.19.8. 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
- **BarStart** - start date of the current (non-daily) bar
- **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
- **EndOfQuarter** - true if next bar will end in a different quarter than this one
- **EndOfWeek** - true if next bar will end in a different week than this one
- **EndOfYear** - true if next bar will end in a different year than this one
- **Event** - user-defined value from Event List file
- **Extra** - unadjusted close (adjusted data), Norgate turnover, or CSV value
- **FunBar** - relative bar number within a multi-bar function calculation
- **FxRatio** - foreign exchange conversion ratio for the current symbol on this bar
- **High or H** - bar high price
- **InXXX** - index constituency flag 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

14.19.9. 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
- **JDSAR** - directional Parabolic SAR (long/short-side stop)
- **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** - MACD signal line = 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
- **SigSpike** - Grimes' Sigma Spike indicator
- **STOC** - stochastics

- **SuperTrend** - Olivier Seban's SuperTrend indicator

14.19.10. 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.

- **Adjusted** - fully back-adjusted price series (splits + dividends) - Data section only
- **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
- **CorrelAvg** - average correlation of a stock to a group of stocks
- **CorrelMed** - median correlation of a stock to a group of stocks
- **CorrelMin** - lowest correlation of a stock to a group of stocks
- **CorrelMinSym** - symbol of stock with lowest correlation to the current stock
- **CorrelMax** - highest correlation of a stock to a group of stocks
- **CorrelMaxSym** - symbol of stock with highest correlation to the current stock
- **CountTrue** - count of all bars for which expression was true
- **DEMA** - double exponential moving average
- **EhlersBP** - bandpass filter as described by John Ehlers
- **EhlersHP** - highpass filter as described by John Ehlers
- **EhlersSS** - super smoother filter as described by John Ehlers
- **EhlersUS** - ultimate smoother filter as described by John Ehlers
- **EMA or XAvg** - exponential moving average
- **ESD** - exponential standard deviation
- **HMA or HAvg** - Hull moving average
- **Highest or HHV** - highest value
- **Hurst** - Hurst exponent (trending vs mean-reverting)
- **KAMA** - Kaufman adaptive moving average
- **Kurtosis** - 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** - 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
- **TEMA** - triple exponential moving average
- **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)
- **VWMA** - volume-weighted moving average
- **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

14.19.11. 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
- **#DataValueFile** - allocate a Data array to receive values from a CSV file rather than a formula
- **#DenseRank** - cross-sectional numeric rank where duplicate (tied) values get the same rank
- **#DVFAlign** - align DataValueFile dates absent from OHLCV bars to the next valid bar date
- **#Fill** - replace NaN values by carrying the last non-NaN value forward
- **#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)
- **#Pad** - replace NaN values with zeros
- **#PercentRank** - cross-sectional percent rank
- **#PercentRankN** - value with specified cross-section percent rank
- **#Rank** - cross-sectional numeric rank
- **#SlowCalc** - forces Data item to not use one-pass calculation (not actually cross-sectional)
- **#StdDev** - cross-sectional standard deviation
- **#Sum** - cross-sectional sum of values

14.19.12. 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.

- **?AccountSize** - value of the AccountSize setting
- **?EndDate** - end date of the script being run
- **?LocalTime** - the current local date and time
- **?OrdersDate** - date for which orders are being generated
- **?RunMode** - mode in which the script is being run
- **?ScriptName** - name of the script being run (without path or extension)
- **?StartDate** - start date of the script being run
- **Abs** - absolute value of a number
- **ArcCos** - determines which angle has the specified cosine
- **ArcSin** - determines which angle has the specified sine
- **ArcTan** - determines which angle has the specified tangent

- **Assert** - require a condition to be true for the script to continue
- **Bound** - limit the range of a value
- **CDF** - cumulative distribution function
- **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
- **DateBars** - count of bars since (or until) a specific date
- **DateDay** - extract the day number from a date
- **DateMonth** - extract the month number from a date
- **DateYear** - extract the year number from a date
- **Days** - number of calendar days between two dates
- **DebugEval** - stop script execution and log the details of a strategy formula's evaluation when a condition is met
- **DllDataCalc** - call a custom external DLL function to calculate a Data item
- **Exp** - exponential function (e^x)
- **Extern** - evaluate expression for a different stock/contract or strategy
- **ICDF** - inverse cumulative distribution function
- **IF** - conditional branch
- **InSet** - whether a value is in a specific set of values
- **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
- **Logit** - inverse logistic function
- **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
- **MissingBars** - number of bars missing before the current bar in this symbol
- **MostRecent** - evaluate expression as of the most recent available bar when one is missing on the current date
- **NoNan** - evaluate an expression with no possibility of a nan (not a number) result
- **NthOfMonth** - whether current bar date is nth specific weekday of current month
- **OrderSum** - evaluates a formula for each non-skipped setup and returns the sum of the values
- **PositionSum** - evaluates a formula for each open position and returns the sum of the values
- **Random** - random number from min to max rounded to step
- **Reason** - return both a value and a string for use in e.g. *ExitLimit*
- **Rescale** - map a value from one scale to another

- **Round** - round value to nearest step
- **Select** - multi-part conditional branch
- **SetupSum** - evaluates a formula for each setup and returns the sum of the values
- **Sign** - sign of a number (1 or -1)
- **Sine** - trigonometric sine of a number of degrees
- **SkipNans** - evaluate a multi-bar function using only non-NaN values within the lookback period
- **Sqr** - square root of a number
- **StratRef** - find the number of a given strategy, and/or allow dynamic Extern strategy references
- **Switch** - value switch function
- **SymRef** - find the number of a given symbol, and/or allow dynamic Extern symbol references
- **Tangent** - trigonometric tangent of a number of degrees
- **TanH** - hyperbolic tangent of a number
- **TargetPrice** - calculate the value of tomorrow's close that would cause an indicator to reach a specific level
- **This** - anchored term in multi-bar function
- **Top** - top N digits of a number
- **UntilEvent** - market days until next event of given type (works past end of imported data)

14.19.13. String Functions

These functions operate on **string values**. Some of them return strings and others return numeric values.

- **Find** - find a string within another string and return its offset
- **Format** - embed numeric or string values within a string using formatting codes
- **FormatDate** - convert a date to a string in any format
- **GetField** - extract a delimited substring from a string
- **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

14.19.14. 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** - The Refinitiv 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

14.19.15. Current Position Information

Information about the current position in a test.

- **BarsHeld** - age of current position in bars
- **Category** - category specified for current position
- **EntryDate** - date on which current position was entered
- **EntryRank** - rank number for this position when EntryScore was evaluated at entry

- **ExitNum** - the number of this exit for this position
- **ExitRank** - rank number for this position based on ExitScore
- **ExitType** - the exit order type for which a *OrderExtra* formula is being evaluated
- **FillFraction** - equity fraction of current position at time of entry
- **FillPrice** - entry or exit fill price of current position
- **FillPriceAvg** - average fill price of current symbol if multiple positions
- **FillPriceMax** - highest fill price of current symbol if multiple positions
- **FillPriceMin** - lowest fill price of current symbol if multiple positions
- **FillQty** - shares or contracts in current position at time of entry
- **FillValue** - value of current position at time of entry
- **FxRatio** - foreign exchange currency ratio for this position on this date
- **IsExit** - whether the formula being evaluated pertains to a position being exited
- **IsOrder** - whether the current stock's setup becomes an order today
- **IsSetup** - whether the current stock is a setup today
- **OrderPrice** - anticipated entry fill price of a position
- **OrderRank** - order rank number for this position's entry across all strategies
- **PrevExitLimit** - previous exit limit price for use in trailing target calculations
- **PrevExitStop** - previous exit stop price for use in trailing stop calculations
- **SetupRank** - rank number by *SetupScore* for this position when it was entered
- **Shares or Contracts** - number of shares or contracts in current position
- **Side** - side of the current position
- **TLValueIn** - the *ValueIn* value for a position that originated from trade list
- **TLValueOut** - the *ValueOut* value for a position that originated from trade list

14.19.16. Current Strategy Information

Information about the current strategy in a test.

- **?Strategy** - name of current strategy
- **?StratType** - type of current strategy
- **StratNum** - ordinal number of current strategy in the script
- **IsTracker** - current strategy is an auto-created **Tracker** for another strategy

14.19.17. Test Statistics Arrays

Used in the special-purpose scripts results.rts and graphs.rts, and can also be used in strategy formulas. Similar to bar fields in that these represent a series of values, one per date.

- **FunStat** - ordinal number of a stat period within the calculation of a multi-bar function
- **S.Alloc** - current allocation amount
- **S.BPx** - strategy bars per period (W, M, Q, Y)

- **S.CashInOut** - cumulative net cash in-out (excluded from backtest stats)
- **S.Collateral** - current total dollar collateral (margin) required to hold open futures, FX, and crypto positions (alias: S.FuturesMargin)
- **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** - total percent of allocation in currently open positions
- **S.FeesInOut** - cumulative net fees in-out (included in backtest stats)
- **S.First** - period number in which first strategy trade entry occurred
- **S.FreeCash** - current free cash balance
- **S.Interest** - net interest received-paid this period
- **S.Invested** - total dollar value of all currently open positions
- **S.LongExp** - total percent of allocation in currently open long positions
- **S.LongInv** - total dollar value of currently open long positions
- **S.LongM2M** - net mark-to-market of currently open long positions
- **S.LossAmt** - dollar (or base currency) P&L of losing trades this period
- **S.LossBars** - duration 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.MarkedToMarket** - strategy mark-to-market flag
- **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.NetDlrMax** - best-case dollar change for this period (open positions marked at their most favorable prices)
- **S.NetDlrMin** - worst-case dollar change for this period (open positions marked at their least favorable prices)
- **S.NetPct** - percent change in allocation value for this period
- **S.NetPctMax** - best-case percent change for this period
- **S.NetPctMin** - worst-case percent change 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.RiskFreeRate** - risk-free rate of return (from RiskFreeRateSym setting)
- **S.RollCosts** - total futures roll costs this period
- **S.Setups** - total count of entry setups this period
- **S.ShortExp** - total percent of allocation in currently open short positions
- **S.ShortInv** - total dollar value of currently open short positions
- **S.ShortM2M** - net mark-to-market of currently open short positions
- **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.TradePctAlloc** - percent P&L of all trades this period based on allocation
- **S.TWEQ** - time-weighted equity value
- **S.Usage** - percent of allocation used by positions held any time this period
- **S.WinAmt** - dollar (or base currency) P&L of winning trades this period
- **S.WinBars** - duration of winning trades this period
- **S.WinPct** - percent P&L of winning trades this period based on position size
- **S.WinPctAlloc** - percent P&L of winning trades this period based on allocation
- **S.Wins** - count of winning exits this period

14.19.18. 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 commission ($)
- **T.CommOut** - exit commission ($)

- **T.DateIn** - date of trade entry (numeric yyyymmdd)
- **T.DateOut** - date of trade exit (yyyymmdd)
- **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.PriceInZ** - theoretical zero-slippage entry price
- **T.PriceOut** - trade exit price
- **T.PriceOutZ** - theoretical zero-slippage 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.RollCosts** - total roll costs paid for a futures trade, in dollars
- **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.Symbol** - symbol 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.

14.19.19. Trade Statistics Functions

These functions can be used in Results, Graphs, Charts, StratData, TestData, TestScan, Library and 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.

A *Count* of 0 is the same as omitting it: all closed trades are included.

Only trades that the test keeps are included. Benchmark trades, including those of a Tracker clone, are kept only when the **KeepTrades** setting includes Benchmark.

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
- **TradeStatKurt** - the kurtosis 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
- **TradeStatSkew** - the skewness 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

14.20. 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.

14.20.1. #Avg

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the average value for all symbols on that date and stores that value in the data item array for every symbol.

Syntax

#Avg *formula*

Example

```
AvgPriceToday: #Avg Close
```

Calculates the average closing price across all stocks today and stores that value in today's slot for each symbol.

14.20.2. #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

```
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.

14.20.3. #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

```
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.

14.20.4. #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

```
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.

14.20.5. #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

```
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.

14.20.6. #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

```
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.

14.20.7. #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

```
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.

14.20.8. #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

```
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.

14.20.9. #Count

Category

Cross-Sectional Functions

Description

For each date, calculates the count of symbols for which a formula can be evaluated on that date and stores that value in the data item array for every symbol.

Syntax

#Count *formula*

Example

```
HaveDataToday: #Count Close
```

Counts how many symbols have a bar for this date and stores that value in today's slot for each symbol.

Notes

If the formula result is **nan** (not a number -- unable to calculate) then that stock is not included in the count.

14.20.10. #DataValueFile

Category

Cross-Sectional Functions

Description

#DataValueFile is a special **Data Section** tag that looks like a cross-sectional function name but is not one.

This tag allocates a Data array to receive its values from a CSV file rather than calculating them with a formula.

Use a **DataValueFile** (with no #) statement in the same Data section to specify the path to the CSV file that contains the data for this item.

Add **#DVFAlign**, **#Fill** and/or **#Pad** to a #DataValueFile item if desired.

Do not provide any type of constant or formula value directly.

Example

```
MyCustomInd: #DataValueFile
MySparseData: #DataValueFile #Pad
```

Your CSV file would need to include columns called *MyCustomInd* and *MySparseData* to populate the specified items.

Notes

See **DataValueFile Items** for a more complete description of this feature.

14.20.11. #DenseRank

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the dense rank of each symbol's value among all symbols on that date (symbols with equal values receive the same rank) and stores those rank numbers in the data item array for each symbol.

Syntax

#DenseRank *formula*

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.

14.20.12. #DVFAlign

Category

Cross-Sectional Functions

Description

#DVFAlign is a special **Data Section** tag that looks like a cross-sectional function name but is not one.

This tag specifies that a **#DataValueFile** item should include values from the CSV file that have dates that are not present in the normal OHLCV bars of the specified symbol, e.g. weekend or holiday dates. Such dates are aligned by changing them to the first valid OHLCV date AFTER the specified date.

Example

```
MySloppyData: #DataValueFile #DVFAlign
```

Keep all values found in this file even if dates are not found in OHLCV bars by changing each such date to the next valid bar date.

Notes

If #DVFAlign is NOT specified then values with misaligned dates are discarded. Select *Log Calculation Details* in Program Options before calculating your Data section to see notifications about any such values.

#Pad or **#Fill** can be specified in combination with #DVFAlign if desired.

See **DataValueFile Items** for a more complete description of this feature.

14.20.13. #Fill

Category

Cross-Sectional Functions

Description

#Fill is a special **Data Section** tag that looks like a cross-sectional function name but is not one.

This tag specifies that a Data item should replace **NaN** values in the calculated data array for this item by carrying the last non-NaN value forward. It works on any Data item, whether an ordinary calculated formula or a #DataValueFile item (for which it carries the last value found in the CSV file forward across bar dates that had no value).

#DVFFill is a backward-compatible synonym for #Fill. These tags were originally supported only for DataValueFile items (as #DVFFill and #DVFPad); since version 2.0.31.6 they work for any Data item, and either spelling may be used with either item type.

Example

```
IndexTrend: #Fill Extern($$SPX, C > MA(C, 200))
```

If the formula cannot be calculated, e.g. for US holiday bars when calculating it for a non-US symbol, the prior value is carried forward.

Notes

If #Fill were not specified, IndexTrend would be NaN on US holiday bars when referencing it from a non-US symbol.

See **#Pad** for an alternative way to replace NaN values.

See **DataValueFile Items** for more about using this tag with a DataValueFile item.

14.20.14. #Highest

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the highest (largest) value for all symbols on that date and stores that value in the data item array for every symbol.

Syntax

#Highest *formula*

Example

```
HighPriceToday: #Highest Close
```

Calculates the highest closing price across all stocks today and stores that value in today's slot for each symbol.

14.20.15. #Lowest

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, calculates the lowest (smallest) value for all symbols on that date and stores that value in the data item array for every symbol.

Syntax

#Lowest *formula*

Example

```
LowPriceToday: #Lowest Close
```

Calculates the lowest closing price across all stocks today and stores that value in today's slot for each symbol.

14.20.16. #Median

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the median value for all symbols on that date and stores that value in the data item array for every symbol.

Syntax

#Median *formula*

Example

```
MedPriceToday: #Median Close
```

Calculates the median closing price across all stocks today and stores that value in today's slot for each symbol.

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.

14.20.17. #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 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).

14.20.18. #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 bars of a given symbol.

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 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).

14.20.19. #Pad

Category

Cross-Sectional Functions

Description

#Pad is a special **Data Section** tag that looks like a cross-sectional function name but is not one.

This tag specifies that a Data item should replace any **NaN** values in the calculated data array for this item with zeros. It works on any Data item, whether an ordinary calculated formula or a #DataValueFile item.

#DVFPad is a backward-compatible synonym for #Pad. These tags were originally supported only for DataValueFile items (as #DVFPad and #DVFFill); since version 2.0.31.6 they work for any Data item, and either spelling may be used with either item type.

Example

```
IndexSignal: #Pad Extern($$SPX, C > MA(C, 200))
```

If the formula cannot be calculated, e.g. for US holiday bars when calculating it for a non-US symbol, the result will be 0 rather than NaN.

Notes

If #Pad were not specified, this IndexSignal would be NaN on US holiday bars when referencing it from a non-US symbol.

See **#Fill** for an alternative way to replace NaN values.

See **DataValueFile Items** for more about using this tag with a DataValueFile item.

14.20.20. #PercentRank

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the percent rank of each symbol's value among all symbols on that date and stores those percent rank values in the data item array.

Syntax

#PercentRank *formula*

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

```
PriceRankToday: #PercentRank Close
```

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.

14.20.21. #PercentRankN

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, calculates the percent rank of each symbol's value among all symbols on that date, determines which symbol has the specified percentile, and stores that symbol's formula value in the data array for every symbol.

Syntax

#PercentRankN(pctl) *formula*

Parameters

pctl - rank percentile (cannot be an expression or formula)

Notes

The pctl argument is a percent rank from 0 (lowest value) to 100 (highest value): #PercentRankN(0) returns the lowest value and #PercentRankN(100) the highest.

Example

```
PriceWith50thPctlToday: #PercentRankN(50) Close
```

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). The selected percentile value is still stored for every symbol, so its stored result is not **nan**.

14.20.22. #Rank

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula for each symbol, then calculates the rank of each symbol's value among all symbols on that date and stores those rank numbers in the data item array for each symbol.

Syntax

#Rank *formula*

Notes

Lowest rank (1) means highest value. Identical values get different rank numbers.

Example

```
PriceRank: #Rank Close
```

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.

14.20.23. #SlowCalc

Category

Cross-Sectional Functions

Description

#SlowCalc is a special **Data Section** tag that looks like a cross-sectional function name but is not one.

Specifying #SlowCalc at the beginning of a Data formula tells RealTest to NOT use the faster **one-pass formula calculation** mode for this formula.

If you have a Data formula that you suspect may be calculating incorrect values, especially around bars with stock splits, try adding this tag to the start of the formula and recalculating it. If the output changes, keep the tag in place, otherwise remove it so the faster calculation speed can remain in use.

See also **Split Handling**, particularly the example "exponential slope" calculation.

14.20.24. #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

```
ReturnDeviation: #StdDev ROC(C, 1)
```

Calculates the standard deviation of 1-day returns across all stocks today and stores that value in today's slot for each symbol.

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.

14.20.25. #Sum

Category

Cross-Sectional Functions

Description

For each date, evaluates a formula and calculates the sum of values for all symbols on that date.

Syntax

#Sum *formula*

Example

```
TotalMktVol: #Sum Volume
```

Calculates the sum of volume across all stocks today and stores that value in today's slot for each symbol.

14.20.26. ?AccountSize

Category

General-Purpose Functions

Description

The value specified in the **AccountSize** Setting or in the UI **Settings Panel**

Notes

It is rarely necessary to refer directly to this setting from a formula in a script.

14.20.27. ?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**.

14.20.28. ?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**.

14.20.29. ?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**.

14.20.30. ?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.

14.20.31. ?EndDate

Category

General-Purpose Functions

Description

The **EndDate** setting value of the current script being run

Notes

The date is returned as both a number (yyyymmdd) and a string (formatted).

For example *Days(?StartDate, ?EndDate)* can be used to count the days of the current test.

If a script includes run-mode-specific **Settings Sections**, the return value will reflect the setting from the section corresponding with the mode the script is running in.

For example, if *ScanSettings* has *EndDate: 2024-01-01* and *TestSettings* has *EndDate: 2020-01-01*, *?EndDate* will return the former if the script was run by pressing *Scan*, or the latter if the script was run by pressing *Test*.

If there is no mode-specific *EndDate* setting for the current run mode, the value from *Settings* is used and returned. If there is no *Settings EndDate* either, the last date from the **Settings Panel** is used.

14.20.32. ?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**.

14.20.33. ?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**.

14.20.34. ?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.

14.20.35. ?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.

14.20.36. ?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**.

14.20.37. ?LocalTime

Category

General-Purpose Functions

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*

14.20.38. ?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**.

14.20.39. ?OrdersDate

Category

General-Purpose Functions

Description

The date for which orders are being generated when a script is run in **Orders Mode**

Example

```
OrderExtra3: Format("{?} 15:59:30 US/Eastern", FormatDate(?OrdersDate,
"yyyy-MM-dd"))
```

This might be used to include a "Good After Time" string in a **template order** by defining a column with *extra3* as its output.

Notes

The date is returned as both a number (yyyymmdd) and a string (formatted according to your *Date Display* choice in **Program Options**).

14.20.40. ?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**.

14.20.41. ?RunMode

Category

General-Purpose Functions

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:

The six names above are the interactive run modes. A seventh value, *parse*, is also possible: it is returned when the script is run in batch parse-only mode (the *-parse* command-line argument), which parses the script and stops.

14.20.42. ?ScriptName

Category

General-Purpose Functions

Description

Name of the script being run

Notes

The string returned is the name of the current script with neither the full folder path nor the .RTS extension.

For example ?ScriptName would be "mr_sample" for c:\RealTest\Scripts\Examples\mr_sample.rts.

14.20.43. ?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.

14.20.44. ?Strategy

Category

Current Strategy Information

Description

Name of the current strategy

Notes

The return value is a **string** containing the full name of the current strategy.

See also **StratNum**, which returns the current strategy's number, and **StratRef**, which allows dynamic strategy lookup.

14.20.45. ?StratType

Category

Current Strategy Information

Description

Type of the current strategy

Notes

The return value is a **string** containing the type of the current strategy.

Possible return values are:

- *Benchmark* - the strategy is a **Benchmark**
- *Combined* - the strategy is a **Combined** section
- *StatsGroup* - the strategy is a **StatsGroup**
- *Strategy* - the strategy is a regular **Strategy**

14.20.46. ?StartDate

Category

General-Purpose Functions

Description

The **StartDate** setting value of the current script being run

Notes

The date is returned as both a number (yyyymmdd) and a string (formatted).

For example *Days(?StartDate, ?EndDate)* can be used to count the days of the current test.

If a script includes run-mode-specific **Settings Sections**, the return value will reflect the setting from the section corresponding with the mode the script is running in.

For example, if *ScanSettings* has *StartDate: 2024-01-01* and *TestSettings* has *StartDate: 2020-01-01*, *?StartDate* will return the former if the script was run by pressing *Scan*, or the latter if the script was run by pressing *Test*.

If there is no mode-specific *StartDate* setting for the current run mode, the value from *Settings* is used and returned. If there is no *Settings StartDate* either, the first date from the **Settings Panel** is used.

14.20.47. ?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 constant **Symbol Reference**, and **SymRef**, 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.

14.20.48. ?Type

Category

Stock/Contract Information

Description

Type 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**.

14.20.49. Abs

Category

General-Purpose Functions

Description

Absolute Value of a number

Syntax

Abs(value)

Parameters

value - formula

Example

```
AbsRet: Abs(ROC(C, 1))
```

Absolute value of the daily return.

14.20.50. 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.

14.20.51. AccountType

Category

Settings

Description

Specifies the type of account being modeled.

Choices

CashSameDay - model a cash account where sale proceeds can be reinvested the same day

CashNextDay - model a cash account where sale proceeds can't be reinvested until the next day

Margin - model a margin account (default)

Notes

This setting affects backtests and order generation in the following ways:

- In the *Margin* account model, **MinFreeCash** defaults to unlimited (can become negative) while in either *Cash* model, *MinFreeCash* defaults to 0 (can't become negative).
- In the *Margin* model, proceeds from any sale made today can be used immediately to buy other positions. In the *CashSameDay* model, proceeds are available the same day only when the sale occurs at the open; proceeds from sales made later in the day (intraday or on the close) are not available until the next day. In the *CashNextDay* model, proceeds from any sale are always delayed one day before they can be used.
- In either *Cash* model, a dividend accrued to an open position cannot fund a new entry on its ex-date, because accrued dividends only become free cash when the position exits, unless **DividendsToCash** is *True*.

14.20.52. Adjustment

Category

Import Specification

Description

Price data adjustment setting

Syntax

Adjustment: *choice*

Choices

Capital - (default) no dividend types are converted to splits

CapitalSpecial - corporate restructuring events (mergers etc.) and special dividends are converted to splits

TotalReturn - all types of dividends 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 and some other sources.

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:

- *Capital* only treats actual stock splits as splits, simulating every other adjustment event (ordinary dividends, special dividends, spin-offs and other corporate restructuring events) as a cash dividend payout. This introduces an occasional large price gap (with corresponding simulated dividend payout) around corporate restructuring events.
- *CapitalSpecial* additionally converts special dividends and corporate restructuring events to splits, which avoids those price gaps, while still correctly simulating payout of ordinary dividends.
- *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. It also puts every lookback price reference and multi-bar indicator on a total-return scale: the current bar's *C* is still the as-traded price, but past bars are scaled down by the accumulated dividend factor. Ratio-based calculations such as *ROC*, *RSI* or a moving average crossover work as intended (they simply measure total return rather than price return), but a formula that compares a past price to an absolute dollar level, e.g. *Lowest(C, 250) > 10*, is testing an adjusted value that can be well below what actually traded.

If you want the realistic fills and dividend accounting of *Capital* but also need a total-return price series (e.g. for momentum or relative-strength ranking), use the **Adjusted** function in your **Data** section rather than importing with *TotalReturn*. That gives you both the unadjusted and the fully-adjusted series in the same test.

Adjustment can optionally be specified separately for each **DataSource** block in an import definition. This enables your data to include a mix of e.g. TotalReturn and Capital-only adjusted symbols. It is even possible to import the same symbol from the same source both ways by defining two *DataSource* blocks for the same source and using the **IncludeList** alias syntax for one of the instances of the symbol.

14.20.53. Adjusted

Category

Multi-Bar Functions

Description

Produce a fully back-adjusted (splits and dividends) price series for use as a **Data section** item

Syntax

Adjusted(price)

Parameters

price - a price expression. In practice only *O*, *H*, *L*, or *C* is meaningful - any expression is accepted, but the result is only correct when the input is one of those raw bar prices.

Notes

This is a *special* function that *only* works as a stand-alone **Data section** item formula. It is calculated in **one pass** over each symbol's bars, walking newest to oldest and applying splits and dividends so that every value is on the same scale as the most recent bar.

The resulting Data item can then be referenced by name in any other formula (Data, Library, Strategy, Scan, etc.) the same way you would reference an OHLC value.

The intended use case is to keep your underlying price data un-dividend-adjusted (e.g.

Adjustment: Capital), so that backtest fills and dividend cash credits behave correctly, while still being able to compute total-return-style factors (momentum, relative strength, etc.) on a fully-adjusted series for ranking.

For this function to incorporate dividends, the imported data must carry dividend values - either from a source that provides them (Norgate, Yahoo) or via a **dividend Event List**. When no dividends are present in the data, the function still applies split adjustment correctly.

Because the calculation walks the full history of each symbol in one pass, the function cannot be embedded inside a larger expression or used outside of the Data section.

Example

```
Data:
 AdjC: Adjusted(C)
 Mom12M: ROC(AdjC, 252)
```

Build a fully back-adjusted close series and use it as the input to a 12-month total-return momentum factor, while the strategy itself continues to fill at the unadjusted O/H/L/C prices.

14.20.54. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Example

```
TrendStrength: ADX(14)
```

ADX > 25 indicates a strong trend; values below 20 suggest a range-bound market.

14.20.55. 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 an 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 **one-pass calculation** when used in the **Data Section**.

Example

```
AdaptMA: AEMA(C, if(ADX(14) > 25, 0.1, 0.04))
```

Faster smoothing (factor=0.1, ~19-bar) in trending markets, slower (factor=0.04, ~49-bar) otherwise.

14.20.56. 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 variable weighting factor.

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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
AdaptSD: AESD(C, if(ADX(14) > 25, 0.1, 0.04))
```

Uses the same adaptive factor logic as AEMA for consistent volatility measurement.

14.20.57. 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**.

14.20.58. AllowMissingBar

Category

Strategy Elements

Description

Whether to allow backtests to trade and orders to be generated after missing bars

Input

True or *False* (default is *True*)

Notes

A "missing bar" is a bar that is absent in this stock while being present in most other stocks from the same exchange.

Often these are caused by temporary "halt" conditions. Missing bars can also simply reflect days when there was no trading volume. (Unless **Padding** is specified, RealTest does not import zero-volume bars.)

By default RealTest WILL allow an entry or an exit in a backtest to occur even if the most recently

completed bar of that symbol is not the prior date in the trading calendar of the exchange (the prior date of most other symbols).

The same rule governs live order generation: in **Orders** mode a symbol whose most recent bar is missing is skipped when this setting is False, so keeping it True (the default) ensures generated orders match a subsequent backtest.

When this setting is False, an entry that would otherwise have occurred on the bar after a missing bar is listed with the other skipped entries in the **Trade List** (when skipped entries are kept) with the skip **Reason** "missing bar".

The setting applies to daily and intraday strategies. Weekly and higher strategies always behave as if it were *False*, since the calendar comparison it is based on has no meaning across their periods.

It is recommended to always use the *True* choice of this setting, which can be done by simply not specifying it. In releases prior to 2.0.31 the default was *False* and the setting is now provided mainly for backward compatibility for older scripts as needed.

See also the general topic of **Calendar Alignment** and the **MissingBars** function, which can be used to count missing bars.

14.20.59. AllowNoVolume

Category

Strategy Elements

Description

Whether to allow backtests to enter and exit positions on bars with zero volume

Input

True or *False* (default is *False*)

Notes

Most data providers including **Norgate** do not provide bars for days on which there were no trades.

The **Padding** setting *AllMarketDays* can be added to an **Import** definition to include those bars.

Bars added as padding always have zero volume and zero price movement (Open, High, Low, Close are all equal to the prior close).

By default RealTest will not allow an entry or an exit in a backtest to occur on such a bar, since in live trading there could have been no such transaction.

If you have a use case that requires transactions on zero-volume bars, add *AllowNoVolume: True* to your **Strategy** definition.

14.20.60. AllowSameName

Category

Settings

Input

True or *False* (default is *False*)

Description

If *True*, RealTest allows items in sections with user-defined named formulas to be replaced by defining a new item with the same name.

If *False* (the default), the parser shows an error message whenever the same name is used twice.

Notes

This applies only to sections with user-defined named formulas, i.e., Data, TestData, StratData,

Results, Graphs, Charts, Trades, Scan, TestScan, and Library. The Parameters section always allows name overrides. Strategy-definition section names (Strategy, Benchmark, StatsGroup, Template) must always be unique.

See **Combining Scripts** for more information about this setting.

14.20.61. AlwaysImport

Category

Settings

Input

True or *False* (default is *False*)

Description

If *True*, RealTest will always run an **Import** (if the script has **defined one**) when any **Run mode** is initiated, prior to running the script in the requested mode.

If *False* (the default), RealTest only runs an *Import* when *Import* was the requested run mode.

Notes

It is not advisable to leave this setting on *True* unless you have a specific need to do so. Importing data can be a time-consuming process. Doing so redundantly defeats the purpose of RealTest's local data file architecture.

14.20.62. 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, and if Close = Open assume the stop was hit, as in Stop mode (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".

14.20.63. ArcCos

Category

General-Purpose Functions

Description

Determines which angle has the specified **Cosine**

Syntax

ArcCos(value)

Parameters

value - a cosine in the range of -1 to 1.

Notes

The return value is expressed in degrees (0 to 180).

To convert degrees to radians, divide by 57.2957795131.

Example

```
Degrees: ArcCos(0.5)
```

Returns 60. All trig functions work in degrees.

14.20.64. ArcSin

Category

General-Purpose Functions

Description

Determines which angle has the specified **Sine**

Syntax

ArcSin(value)

Parameters

value - a sine in the range of -1 to 1.

Notes

The return value is expressed in degrees (-90 to 90).

To convert degrees to radians, divide by 57.2957795131.

Example

```
Degrees: ArcSin(0.5)
```

Returns 30.

14.20.65. ArcTan

Category

General-Purpose Functions

Description

Determines which angle has the specified **Tangent**

Syntax

ArcTan(value)

Parameters

value - a tangent

Notes

The return value is expressed in degrees (-90 to 90).

To convert degrees to radians, divide by 57.2957795131.

Example

```
Degrees: ArcTan(1)
```

Returns 45.

14.20.66. Arg1-Arg9 / Ref1-Ref9

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.

Library function arguments can be either numbers or strings (or formulas that evaluate to either).

Arg1 through *Arg9* are passed *by value*: each is evaluated once, at the bar where the library function is referenced, and passed in as a single value. For example *myLibFunc(Close)* evaluates *Close* to the current close and passes that one number. A by-value argument is a constant, not an array, so it should not be given a bar offset such as *Arg1[1]* or passed to a multi-bar function.

Ref1 through *Ref9* are those same arguments passed *by reference*: *Ref1* is the first argument, *Ref2* the second, and so on, but each is re-evaluated in the calling formula's context every time it is used. A by-reference argument therefore behaves as an array -- *Ref1[1]* returns the prior bar's value, and an argument can be used in a multi-bar function such as *MA(Ref1, 20)*.

A library formula may freely mix the two forms, even for the same argument -- use *ArgN* where you want the fixed value at the call bar and *RefN* where you want the full series.

For example, a library function defined as *myMA: MA(Ref1, Arg2)* and invoked as *myMA(Close, 20)* computes a true 20-bar average of the close: *Ref1* supplies the close as a series while *Arg2* supplies the constant length 20. Using *Arg1* in place of *Ref1* would instead average 20 copies of the current close (i.e. just return the close). By-reference gives the same result as by-value whenever an argument does not vary by bar, and differs only when it does.

If an argument expression evaluates to n/a at the call bar, the call does not fail as a whole: any *ArgN* read of that argument sees n/a, while *RefN* uses are unaffected, since a by-reference argument is only evaluated where it is actually used. (Through RealTest 2.0.32.1, one n/a argument caused the entire call to return n/a even if the argument was never read by value.)

14.20.67. Assert

Category

General-Purpose Functions

Description

Require a condition to be true for the script to continue

Syntax

Assert(condition, text {optional})

Parameters

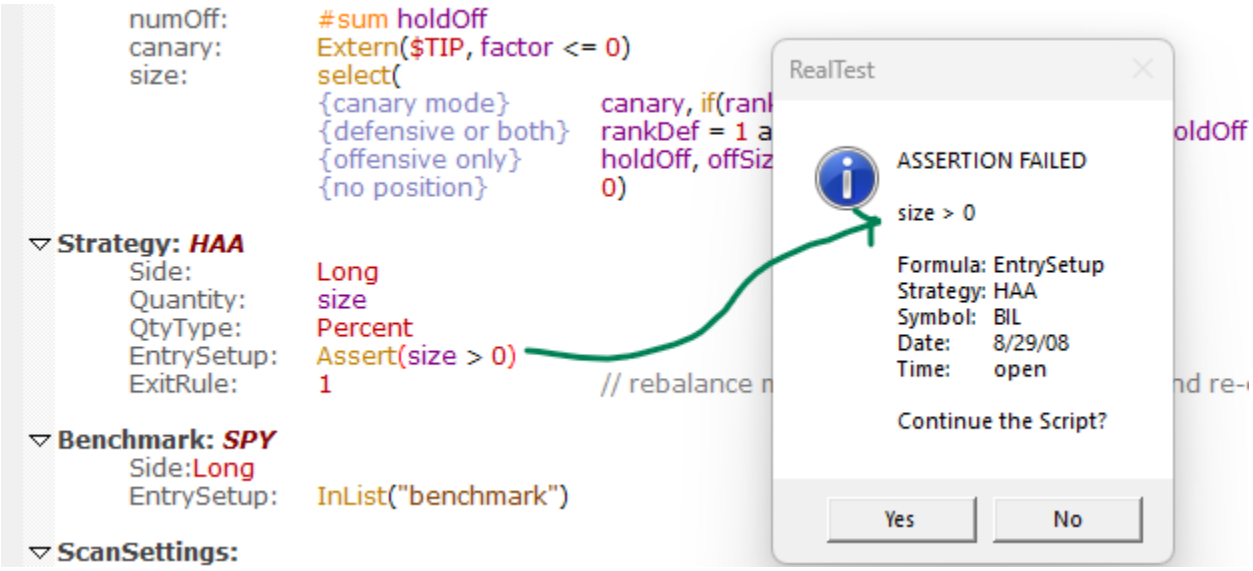
condition - formula specifying a condition

text - formula specifying text to display in the popup message

Notes

When the condition is true, the function does nothing.

When the condition is false, a pop-up message appears:



The default text shows the assertion that failed and which formula, strategy, symbol, date and time the formula was being evaluated for.

If the optional *text* parameter is provided, that text is displayed below the default text.

The script can either be continued (Yes) or stopped (No) after the pop-up is dismissed.

Example

```
EntrySetup: Assert(C > 0) and RSI(2) < 10
```

Pops up a message and pauses if Close is not positive.

14.20.68. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Examples

```
MyATR: ATR(14)
```

Pre-calculating ATR in the Data section improves backtest speed.

```
ExitStop: C - 2 * MyATR
```

A volatility-adjusted trailing stop using ATR.

14.20.69. 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 is of non-daily **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.

Example

```
After2024: BarDate >= 20240101
```

Dates are numeric in YYYYMMDD format.

14.20.70. 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.

14.20.71. 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**.

Example

```
ExitRule: BarsHeld >= 10
```

Exit after holding for 10 bars. BarsHeld is 0 on entry day, 1 after the first night, etc.

14.20.72. 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.

Example

```
NearEnd: BarsLeft < 5
```

Useful for avoiding new entries near the end of available data or a contract's expiration.

14.20.73. BarSize

Category

Settings or **Strategy Elements** or **Data Section**

Description

Specifies a bar size (periodicity). Its scope depends on where it appears - in the Settings section, in a Strategy, or in the Data section.

Notes

In Settings - sets the default bar size for the whole script, used for scans as well as tests. If a

Settings bar size is not specified, the value from the **Settings Panel** is used.

In a Strategy - a **Strategy** can run at its own bar size, different from the Settings bar size, enabling mixed-timeframe tests.

In the Data section - sets the bar size for a group of data items, or for an individual item via a bar-size name prefix; Data arrays are independent of strategy bar sizes. See **Bar-Size-Specific Items** for the two ways to do this.

Valid values are *Daily*, *Weekly*, *Monthly*, *Quarterly*, *Yearly*.

See **Bar Sizes and Multiple Timeframes** for how mixed bar sizes work across these contexts.

14.20.74. 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.

14.20.75. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
LowerBand: BBBot(20, 2)
```

14.20.76. BBBotF

Category

Multi-Bar Functions

Description

Bollinger band bottom as a function

Syntax

BBBotF(expr, len, mult)

Parameters

expr - data series formula

len - lookback period

mult - number of standard deviations

Notes

The **BBBot** 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolBot: BBBotF(V, 20, 2)
```

Unlike BBBot (which uses Close), BBBotF calculates bands for any data series.

14.20.77. 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 - BBBot(len, mult)) / (BBTop(len, mult) - BBBot(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 **one-pass calculation** when used in the **Data Section** with a non-variable

count.

Example

```
BPct: BBPct(20, 2)
```

14.20.78. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolPct: BBPctF(V, 20, 2)
```

Unlike BBPct (which uses Close), BBPctF calculates %B for any data series.

14.20.79. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
UpperBand: BBTop(20, 2)
```

14.20.80. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolTop: BBTopF(V, 20, 2)
```

Unlike BBTop (which uses Close), BBTopF calculates bands for any data series.

14.20.81. 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:

lower = ABS(BBBot(len1, mult) - BBBot(len2, mult))
upper = ABS(BBTop(len1, mult) - BBTop(len2, mult))
BBTrend = (lower - upper) / Avg(C, 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
Trend: BBTrend(20, 50, 2)
```

Positive values indicate uptrend; negative values indicate downtrend.

14.20.82. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolTrend: BBTrendF(V, 20, 50, 2)
```

Unlike BBTrend (which uses Close), BBTrendF calculates trend for any data series.

14.20.83. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
BW: BBWidth(20, 2)
```

Low BBWidth values indicate a "squeeze" that may precede a breakout.

14.20.84. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolWidth: BBWidthF(V, 20, 2)
```

Unlike BBWidth (which uses Close), BBWidthF calculates width for any data series.

14.20.85. Benchmark

Category

Script Sections

Description

Defines a benchmark strategy

Notes

See **Special Strategy Types** for details.

By default, benchmarks process their entry setups ahead of all strategies (each in its own entry phase, in declaration order); see **EntryPhase**.

14.20.86. 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

Example

```
BoundedRSI: Bound(RSI(14), 20, 80)
```

Clips RSI to the 20-80 range.

14.20.87. 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 needed when a multi-strategy script includes strategies for different countries or for markets with different trading calendars. It is also how a strategy gets a 24/7 calendar: if the referenced symbol is a crypto-typed symbol (SymInfo Type starting with "cry"), order dates advance over weekends on a 7-day grid, as live order generation against 24/7 venues requires.

A 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.

Do not prefix the symbol with $ -- that is only required for symbol references within formulas.

Within a formula, the CalendarSym of the current strategy can itself be referenced using *Extern($CalSym, expression)* -- see **Extern**.

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.)

See also the general topic of **Calendar Alignment**.

14.20.88. CashInOut

Category

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*.

Unlike **FeesInOut**, *CashInOut* is NOT included in backtest stats such as the annual return, max drawdown, and daily net percent gain or loss.

Examples

Add $1,000 every month at the start of the month:

```
CashInOut: if(Month <> Month[1], 1000, 0)
```

Withdraw 40% of last year's net profit at the start of each year:

```
CashInOut: if(Year[-1] > Year, -0.4 * max(0, Combined(Sum(S.NetDlr, S.BPY))), 0)
```

See also the **annual_taxes.rts** example script.

14.20.89. 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 annual risk-free rate as determined by the 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).

14.20.90. CashList

Category

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*.

Unlike **FeesList**, items in *CashList* are NOT included in the stats calculations of a backtest, such as the annual return, max drawdown, and daily net percent gain or loss.

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.	

14.20.91. Category (definition)

Category

Strategy Elements

Description

Assigns a category value to a position at entry time

Input

Formula expression

Notes

This formula is evaluated immediately after 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.

14.20.92. 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**.

14.20.93. 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.

Example

```
MyCCI: CCI(20)
```

14.20.94. CDF

Category

General-Purpose Functions

Description

Cumulative Distribution Function

Syntax

CDF(value)

Parameters

value - formula

Notes

The CDF calculates the probability that a random observation will be less than or equal to a certain value. By normalizing financial data to a range of 0 to 1, it enables consistent comparison across assets and periods, aiding in the creation of trading indicators.

Example

```
NormProb: CDF((C - MA(C, 20)) / StdDev(C, 20))
```

Probability (0 to 1) that a normally distributed value would be at or below the current z-score.

14.20.95. Chart Settings

Category

Settings or **Scan Settings** or **Order Settings**

Description

Settings that control chart appearance and behavior from a script

Notes

Each of these settings controls the same chart option as the **chart window** options dialog. Setting one from a script has exactly the effect of changing it in the dialog: it becomes the current chart option for the session - applied both to chart images saved via **SaveScanChartsTo / SaveTradeChartsTo** and to chart windows opened interactively after the script runs - and persists as the saved default thereafter. This makes chart output from command-line runs deterministic without needing a hand-maintained realtest.ini.

The available settings, in alphabetical order:

- **ChartBarSize** (Daily, Weekly, Monthly, Quarterly or Yearly) - the bar size charts display
- **ChartCandlestick** (True/False) - draw candlesticks rather than OHLC bars
- **ChartEntryExitLine** (True/False) - draw a line connecting a trade's entry and exit on trade charts
- **ChartGrid** (True/False) - show grid lines

- **ChartIndicatorPane** (True/False) - show the upper indicator pane(s)
- **ChartLineWidth** (1 to 9) - indicator line width in pixels
- **ChartLogScale** (True/False) - use a logarithmic price scale
- **ChartShowOtherTrades** (True/False) - also draw the symbol's other trades that fall within the visible date range of a trade chart
- **ChartSplitAdjust** (True/False) - show split-adjusted prices
- **ChartTradeExtra** (bar count) - extra bars a trade chart shows beyond the trade
- **ChartTradeExtraAfter** (Entry or Exit) - whether a trade chart ends ChartTradeExtra bars after the trade's entry or its exit (the "After" choice in the dialog; default is Exit)
- **ChartTradeLabels** (None, Selected or All) - which trade markers get text labels on a trade chart: none, only the chart's own trade, or every trade shown (the default); see **chart windows** for how trades are marked
- **ChartVisibleBars** (bar count) - minimum number of bars visible; a trade chart widens beyond this as needed to fit its whole trade
- **ChartVolumePane** (True/False) - show the volume (lower) pane

ChartImageWidth and **ChartImageHeight** (200 to 8000) set the pixel size of the chart images saved by SaveScanChartsTo and SaveTradeChartsTo. Unlike the options above, they apply only to the script they appear in and do not affect interactive chart windows. When they are not specified, saved images come out at the size of the most recently sized chart window (as remembered in realtest.ini), or 1600 x 1000 if no chart window has ever been sized.

Example

```
Settings:
   SaveTradeChartsTo: ?output?\charts
   SaveTradeChartsMax: 50
   ChartVisibleBars: 30
   ChartTradeExtra: 10
   ChartEntryExitLine: True
   ChartShowOtherTrades: True
   ChartImageWidth: 1200
   ChartImageHeight: 800
```

Chart options for an already-open chart window can still be changed interactively via the **Chart Menu** options dialog.

14.20.96. Charts

Category

Script Sections

Description

Chart indicator definitions

Notes

See **Charts Section** and **Candlestick/Bar Charts**.

14.20.97. 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.au - ASX 200 industry indices

$XKO.au - 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(&cii**, expression). The equivalent numeric form *Extern(&99, expression)* also works.

14.20.98. 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).

14.20.99. 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 Refinitiv 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 #ByGroup factor*.

14.20.100. 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.

Example

```
PrevClose: C[1]
```

Use [offset] to reference previous bars. C[1] is yesterday's close, C[5] is 5 bars ago.

14.20.101. 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.

14.20.102. Collateral / FuturesMargin

Category

Strategy Elements

Description

Per-contract cash collateral (margin) required to hold a futures, FX, or crypto position

Input

Formula specifying dollars per contract of required collateral

Notes

Collateral was formerly named *FuturesMargin*, which is still accepted as an alias.

Collateral is the cash per contract that a broker requires in order to hold a position in an instrument that is traded on margin rather than paid for in full -- typically **futures, FX, and crypto**. Such a position never "uses margin" in the sense of borrowing funds from the broker; the collateral is simply cash that must be set aside to hold the position.

Brokers typically do not pay interest on the cash used as collateral and do not allow a position to be opened if there is insufficient cash to cover the requirement.

The presence of a *Collateral* formula in a strategy also acts as a **switch** that changes how **free cash** is calculated for that strategy. When a position is collateralized, only the collateral (contracts × *Collateral*) ties up cash, rather than the full notional value. Just as important, a **short** collateralized position does **not** add its sale proceeds to free cash -- unlike a short stock position, whose proceeds increase free cash. This in turn affects the **MinFreeCash** constraint, **CashIntPct** interest, and the **S.FreeCash** and **S.Collateral** statistics.

Historical data typically does not include historical margin/collateral requirements. **Norgate** provides current futures margins, which RealTest imports and stores in **InfoMargin**. Users of other data sources can provide current requirements in their **SymInfoFile** if desired.

If *Collateral* is not specified in a strategy then *InfoMargin* is used by default. Provide a *Collateral* formula if you want to model approximate historical requirements, e.g. based on ATR.

If a strategy does not provide *Collateral* and a symbol does not have a value for *InfoMargin* then RealTest treats that symbol as an ordinary, fully-paid instrument for *CashIntPct* and *MinFreeCash* purposes.

The total collateral in effect on each date of a test can be accessed via the **S.Collateral** stats array.

14.20.103. 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**.

Example

```
CombEquity: Combined(S.Equity)
```

Access the combined equity curve of all strategies.

14.20.104. Combined (section)

Category

Script Sections

Description

Allows definition of all-strategy combined constraints such as **MaxExposure**, **MaxInvested**, **MaxPositions**, etc.

Notes

See **Special Strategy Types** for details.

14.20.105. CommCurrency

Category

Settings

Description

Specifies how the **Commission** formula is interpreted in a multi-currency script.

Choices

Instrument - commission values are interpreted as being in the currency of the instrument being traded (default)

Account - commission values are interpreted as being in the base currency of the account

Notes

Some brokers, including IB (Interactive Brokers), charge trade commissions in the currency of the instrument being traded. Other brokers may charge commissions in the account's base currency. This setting lets you tell RealTest which way your broker does it.

CommCurrency also determines how commission values in **Imported Trade Lists** are interpreted.

See also **Testing Multi-Currency Strategies**.

14.20.106. Commission

Category

Strategy Elements

Description

Commission amount, in instrument or account currency units, for each trade

Input

Formula specifying a commission 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: *Min(0.01 * FillValue, Max(0.005 * Shares, 1)).*

IB's Canadian commission formula is: *Min(0.005 * FillValue, Max(0.01 * 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 * Max(6.00, 0.0008 * 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 be less than $0.005/share.

By default, Commission is assumed to be in the currency of the instrument being traded. If your broker (unlike IB) charges a commission rate in local account currency regardless of the instrument's currency, use the **CommCurrency** setting to specify *Account*.

14.20.107. 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**) (or the strategy's own *S.Equity* when **StartPercent** is specified) 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**.

14.20.108. 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. A *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: $BCC* to your import and refer to *InBCC* in your formulas to determine whether and when a stock was trading as a "Special Purpose Acquisition Company".

See **Event List Files** 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.

14.20.109. Correl

Category

Multi-Bar Functions

Description

Pearson's Correlation of two series

Syntax

Correl(expr1, expr2, count)

Parameters

expr1 - data series formula

expr2 - data series formula

count - lookback period

Notes

Correl(ROC(C,1), Extern($SPY, ROC(C,1)), 100) would be a simple way to calculate the rolling 100-day correlation of a stock to SPY.

See also **Spearman**, **CorrelAvg**, etc.

Example

```
SpyCorr: Correl(ROC(C, 1), Extern($SPY, ROC(C, 1)), 100)
```

100-day rolling correlation of daily returns to SPY.

14.20.110. CorrelAvg

Category

Multi-Bar Functions

Description

Average correlation of a stock to a group of stocks

Syntax

CorrelAvg(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelAvg(InDJI, ROC(C,1), 20) would return the average 20-bar correlation of each DJIA constituent to the current stock.

See also **Correl**, **Spearman**, **CorrelMed**, **CorrelMin**, **CorrelMinSym**, **CorrelMax**, **CorrelMaxSym**

Example

```
AvgCorr: CorrelAvg(1, ROC(C, 1), 20)
```

Average 20-day return correlation to all other stocks in the data set.

14.20.111. CorrelMed

Category

Multi-Bar Functions

Description

Median correlation of a stock to a group of stocks

Syntax

CorrelMed(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelMed(InDJI, ROC(C,1), 20) would return the median 20-bar correlation of each DJIA constituent to the current stock.

See also **Correl**, **Spearman**, **CorrelAvg**, **CorrelMin**, **CorrelMinSym**, **CorrelMax**, **CorrelMaxSym**

Example

```
MedCorr: CorrelMed(1, ROC(C, 1), 20)
```

Median is more robust to outlier correlations than average.

14.20.112. CorrelMin

Category

Multi-Bar Functions

Description

Lowest correlation of a stock to a group of stocks

Syntax

CorrelMin(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelMin(InDJI, ROC(C,1), 20) would return the lowest 20-bar correlation of any DJIA constituent to the current stock.

See also **Correl**, **Spearman**, **CorrelAvg**, **CorrelMed**, **CorrelMinSym**, **CorrelMax**, **CorrelMaxSym**

Example

```
MinCorr: CorrelMin(1, ROC(C, 1), 20)
```

14.20.113. CorrelMinSym

Category

Multi-Bar Functions

Description

Symbol of stock with lowest correlation to the current stock

Syntax

CorrelMinSym(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelMinSym(InDJI, ROC(C,1), 20) would return the symbol of the DJIA constituent with the lowest 20-bar correlation to the current stock.

The symbol is returned as both a number and a string. Use **SymRef** to refer to symbol with **Extern**.

See also **Correl**, **Spearman**, **CorrelAvg**, **CorrelMed**, **CorrelMin**, **CorrelMax**, **CorrelMaxSym**

Example

```
LeastCorr: CorrelMinSym(1, ROC(C, 1), 20)
```

Returns a symbol value; use with SymRef and Extern to reference that stock's data.

14.20.114. CorrelMax

Category

Multi-Bar Functions

Description

Highest correlation of a stock to a group of stocks

Syntax

CorrelMax(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelMax(InDJI, ROC(C,1), 20) would return the highest 20-bar correlation of any DJIA constituent to the current stock.

See also **Correl**, **Spearman**, **CorrelAvg**, **CorrelMed**, **CorrelMin**, **CorrelMinSym**, **CorrelMaxSym**

Example

```
MaxCorr: CorrelMax(1, ROC(C, 1), 20)
```

14.20.115. CorrelMaxSym

Category

Multi-Bar Functions

Description

Symbol of stock with highest correlation to the current stock

Syntax

CorrelMaxSym(filter, expr, count, spearman {0}, log {0})

Parameters

filter - stock filter formula

expr - value to correlate

count - lookback period

spearman {0} - use Spearman's Rank Correlation method (default is to use Pearson's Correlation)

log {0} - log the individual correlations when the formula is evaluated

Notes

CorrelMaxSym(InDJI, ROC(C,1), 20) would return the symbol of the DJIA constituent with the highest 20-bar correlation to the current stock.

The symbol is returned as both a number and a string. Use **SymRef** to refer to symbol with **Extern**.

See also **Correl**, **Spearman**, **CorrelAvg**, **CorrelMed**, **CorrelMin**, **CorrelMinSym**, **CorrelMax**

Example

```
MostCorr: CorrelMaxSym(1, ROC(C, 1), 20)
```

Returns a symbol value; use with SymRef and Extern to reference that stock's data.

14.20.116. 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).

Example

```
CosVal: Cosine(60)
```

Returns 0.5. Parameter is in degrees.

14.20.117. 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).*

Example

```
MACross: Cross(MA(C, 5), MA(C, 20))
```

True if the 5-day MA was below the 20-day yesterday and above it today.

14.20.118. 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 **one-pass calculation** when used in the **Data Section** without a count argument.

14.20.119. 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.

Example

```
MyCRSI: CRSI(100, 3, 2)
```

Extreme low values (e.g., < 10) indicate oversold conditions.

14.20.120. 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 *CSVDateFmt* is to use the *Date Input* 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.

14.20.121. 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*.

14.20.122. CSVFields

Category

Import Specification

Description

CSV field order (comma-separated list)

Choices

Symbol - symbol name -- only applicable when CSVFile (single-file / multi-symbol CSV) is used; with DataPath, each symbol comes from its data file's name

Date - bar date

Time - bar time

Open - open price

High - high price

Low - low price

Close - close price

Volume - volume

AdjClose - adjusted close price

RealClose - unadjusted (as-traded) close price

Split - split ratio (real close / adjusted close)

Dividend - dividend amount (should be 0 except on ex-dividend dates)

Extra - value that can be referenced as **Extra**

Notes

CSV field order **does not need to** be specified for **CSV Import** - if omitted, RealTest auto-detects the column layout from the header row (by names such as Date, Open, High, Low, Close, Volume). Specify CSVFields when the files have no header row or use non-standard column names.

A CSVFields statement applies to the **DataPath** and **CSVFile** statements that follow it, and carries forward to subsequent DataSources as well, until another CSVFields statement replaces it. Each CSVFields statement fully replaces the prior mapping rather than adding to it. Any DataSource that no CSVFields mapping applies to auto-detects its columns from its own file's header row.

CSVFields: Auto stops a prior mapping from carrying forward and restores header auto-detection for the current DataSource and the ones after it - useful when an earlier CSV source declared explicit fields but a later source (such as **CSIFutures**) should auto-detect from its own headers.

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".

An empty or "nan" Open, High or Low cell is treated as missing and filled from that bar's Close (noted in the import log); a bar whose Close cell is empty or "nan" is dropped. A literal 0 is imported as-is (see **RepairData**). A backtest never fills at a NaN or zero price: such an entry is skipped with the reason "bad price", and no exit is taken on a bar with a NaN or zero price field.

Providing split adjustment data

RealTest needs a cumulative split factor for each bar in order to correctly perform **on-the-fly split adjustment** when calculating multi-bar functions and indicators.

There are three alternative ways to provide this information in a CSV file. Choose the one that matches your data source.

- **AdjClose** - Use when your OHLCV data is unadjusted (as-traded prices) and your data source provides a separate adjusted close column. RealTest derives the split factor as Close / AdjClose. This is the format used by Tiingo.
- **RealClose** - Use when your OHLCV data is split-adjusted and your data source provides a separate unadjusted close column. RealTest derives the split factor as RealClose / Close, then un-adjusts the OHLCV data back to as-traded prices. This is the format used by Norgate.
- **AdjClose + RealClose** - If your data source provides both an adjusted and an unadjusted close, you may specify both. RealTest derives the split factor as RealClose / AdjClose and un-adjusts the OHLCV data. The OHLCV columns are assumed to be split-adjusted in this case (same as RealClose alone).
- **Split** - Use when your data source provides the split ratio directly instead of a second close price. OHLCV data must be unadjusted. Two formats are supported and auto-detected:

  1. Event-based ratio: The split ratio on ex-split dates (e.g. 2 for a 2:1 split, 0.5 for a 1:2 reverse split), with 1.0 on all other dates. Dates must be in descending order.
  2. Cumulative factor: The cumulative split adjustment factor on every bar (e.g. 1.0 for recent bars, 2.0 for all bars before a 2:1 split). Dates may be in any order.

Split cannot be combined with AdjClose or RealClose - it is an alternative way to provide split

information, not a supplement to those columns.

If none of these columns are specified, RealTest assumes the data is unadjusted with no split history, and all split factors default to 1.0.

14.20.123. 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. If the file has no header row, or its header uses non-standard column names, a **CSVFields** declaration must specify each column's location. When the header uses recognized names, the columns are detected automatically.

For multi-file CSV import (one file per symbol), use **DataPath** instead of *CSVFile*.

See **CSV Import** for additional details.

14.20.124. 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.

14.20.125. 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.

14.20.126. Data

Category

Script Sections

Description

Named formulas calculated for each bar of each symbol and stored in memory arrays before tests or scans are run

Notes

See **Data Section** for a more detailed description.

14.20.127. 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.

14.20.128. DataPath

Category

Import Specification

Description

CSI Futures, CSI Stocks, CSV, or MetaStock file location

Input

The full path to a directory (folder) that contains a collection of local data files.

Notes

By default all data files found in each specified *DataPath* are included in the import.

Specify one or more **IncludeList** to optionally filter the files to include only those symbols listed.

IncludeList can also be used to provide **InList** names or numbers with local-file import.

Specify **KeepNonIncluded: True** if you want to define *InList* names while still importing all the data files.

See **CSI Futures Import**, **CSI Stocks Import**, **CSV Import**, or **MetaStock Import** for further details.

14.20.129. 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

EODHD - import stock data from **eodhd.com**

MetaStock - import stock data from a **local MetaStock** database

CSIFutures - import futures data from CSI Unfair Advantage portfolios

CSIStocks - import stock data from CSI Unfair Advantage portfolios

CSV - import any type of data from your own **local CSV files**

14.20.130. 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.

Example

```
AdjVol: DataType(V * C, 2)
```

Force the result to be treated as volume (type 2) for correct split adjustment.

14.20.131. DataValueFile

Category

Data Section

Description

Specifies the path of a CSV file containing data to populate the values of one or more Data Section items

Example

```
Data:
   DataValueFile: ?scriptpath?\my_custom_data.csv
   MyData1: #DataValueFile
   MyData2: #DataValueFile
```

Processes my_custom_data.csv when Data is recalculated.

This CSV file would contain the columns: Date,Symbol,MyData1,MyData2.

That header row would be followed by a row for every date+symbol pair for which you have data.

Notes

A script can specify any number of DataValueFile paths. Each such file is read when the Data section needs recalculation.

See **DataValueFile Items** for a complete description of this feature.

14.20.132. 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**.

Example

```
RefDate: Date(2024, 1, 1)
```

Returns 20240101. Can be compared with BarDate or passed to DateBars.

14.20.133. DateBars

Category

General-Purpose 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

`ROC(C, DateBars(Date(2016,2,1)) + 1)`

Returns the percent gain or loss since February 1, 2016.

(The +1 is needed to make the bar count *inclusive* of the specified bar when using the count as a *length* argument.)

`C[DateBars(Date(Year,1,1))]`

Returns the closing price for the first trading date of the current year.

(Here the bar count is an *offset* rather than a *length* so the +1 is not needed.)

14.20.134. 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.

14.20.135. DateDisplay

Category

Settings

Description

Specifies how dates are displayed in the user interface, scans, trade lists, etc.

Notes

This setting optionally overrides the default date display format as defined in **Program Options**.

The override applies while the script is running and the prior setting is restored when it finishes.

See **Date Format Pictures** for details about the format specification string.

14.20.136. DateInput

Category

Settings

Description

Specifies how dates are parsed when it isn't clear whether the first two numbers are day/month or month/day.

Notes

RealTest can parse dates in any known format without being told what the format is. The only exception is when day and month precede year. In this case it is not always clear which is which, so a setting is required.

This setting optionally overrides the default date input format as defined in **Program Options**.

Note that this setting is independent of the **DateDisplay** format, which is not used when parsing dates in scripts or input files.

14.20.137. 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.

14.20.138. DateSym

Category

Settings

Description

Symbol to use to interpret "Earliest" or "Latest" in **StartDate** or **EndDate**

Notes

By default *Earliest* means the oldest date and *Latest* the newest date of any symbol in the current data file.

If *DateSym* is specified then that symbol is used instead.

A good use of *DateSym* is to avoid the common problem where adding a currency symbol like AUDUSD to your data file causes the "latest" bar to be one for which no stocks have bars.

Do not prefix the symbol with $ -- that is only required for symbol references within formulas.

14.20.139. 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.

14.20.140. 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.

14.20.141. 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.

Example

```
IsMonday: DayOfWeek = 1
```

1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday.

14.20.142. 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.

14.20.143. 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.

Example

```
CalDays: Days(BarDate[20], BarDate)
```

Calendar days between 20 bars ago and today. Pass 1 as third argument for market days.

14.20.144. DaysPerYear

Category

Settings

Description

Number of market days per year for stats purposes

Notes

This becomes the value returned by **S.BPY** for Daily **BarSize** tests (S.BPY is logically always 52 for Weekly, 12 for Monthly, 4 for Quarterly, 1 for Yearly).

The default if not specified is 252.

Use 365 for a 7-day asset such as Crypto.

S.BPY is used internally to calculate the following:

- S.BPW = S.BPY / 52

- S.BPM = S.BPY / 12

- S.BPQ = S.BPY / 4

- HVOL(len) = StdDev(log(c/c[1]), len) * 100 * Sqr(S.BPY)

S.BPY is also used in some of the formulas in Graphs.rts and Results.rts.

14.20.145. DividendsToCash

Category

Settings

Description

Specifies whether dividends accrued to open positions count as free cash from the ex-dividend date rather than when the position exits.

Choices

True - accrued dividends are free cash from the ex-date

False - accrued dividends become free cash when the position exits (default)

Notes

This setting is only relevant when the imported data includes dividends as per-share amounts on ex-dividend dates (**Adjustment***: Capital*, the default for most data sources) and the strategy does not set **IgnoreDividends**.

A dividend accrues to the open position on its ex-date. From that day it is part of the position's open profit and of test equity. By default it does not become free cash until the position closes, when it is realized as part of the trade's profit.

With *DividendsToCash: True*, the accrued amount is added to free cash on the ex-date, so **MinFreeCash**, the *Cash* models of **AccountType**, **S.FreeCash** and the **CashIntPct** /

MarginIntPct interest calculations all see the dividend immediately. Equity, trade profits and all other statistics are unchanged either way. Under a *Cash* account model, the dividend is not treated as sale proceeds when the position later exits, so free cash does not drop on the exit day.

The setting applies to **Compounded** strategies only. A non-compounded strategy sizes from its starting allocation and ignores closed profits, and accrued dividends are treated the same way.

Dividend pay dates are not part of the imported data, so the credit is always on the ex-date, which is somewhat earlier than a real account receives the cash.

A typical use is booking a spin-off. Norgate records a spin-off as a capital-adjustment distribution, i.e. a cash dividend on the parent for the value of the shares received, and a **TradeList** row can then buy the child shares on the ex-date for exactly that amount. In a cash account that is fully invested, the child entry is skipped for lack of free cash unless *DividendsToCash* is *True* (or the trade list is played back before its *TLEndDate*, which bypasses the free cash check).

14.20.146. 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-blank string or a non-zero numeric value then whatever it returns is added to the **test debug or log output**.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (see also the debug_and_validate.rts example in Examples-New\12_special_topics):

```
DebugEntry: if(debug_sym and long_setup, Format("limit={#2}, nextlow={#2}, miss={#2}", LongLimit, next_low,  next_low - LongLimit), "")
```

14.20.147. DebugEval

Category

General-Purpose Functions

Description

Stop script execution and log the details of a strategy formula's evaluation when a condition is met

Syntax

DebugEval(condition, formula)

Parameters

condition - when to stop execution and evaluate the formula

formula - the formula to evaluate

Notes

DebugEval is meant to be wrapped around a strategy formula when necessary to learn the details of how it is being calculated in the context of a specific point in a running test.

For example say you have a complex Quantity formula such as:

```
Quantity:        Select(curqty = 0 and CanEnter and newqty > 0, newqty,
                 curqty > 0 and mustexit, 0,
                 curqty > 0 and rebalday and abs(pctdiff) > rebaldiff, newqty)
```

(from the *clenow_stocks_on_move_dynamic.rts* example script)

You want to see exactly how it is being calculated for AAPL on 2024-06-21:

```
Quantity:        DebugEval(Symbol=$AAPL and BarDate = 20240621,
                 Select(curqty = 0 and CanEnter and newqty > 0, newqty,
                 curqty > 0 and mustexit, 0,
                 curqty > 0 and rebalday and abs(pctdiff) > rebaldiff, newqty))
```

Run the test and it will run as normal until coming to that symbol and date, then will:

- turn on the same mechanism used for "Show Evaluation" option in the **Debug Panel**
- evaluate the formula, causing full evaluation details to be written to a log window
- abort the test

At this point you can examine the log output and also use the debug panel as needed to explore data values in memory, open positions, etc.

Note that *DebugEval* cannot be used in Data or Scan formulas. This is because those are calculated in separate threads and this function is not reentrant. It is intended for use in Strategy formulas only.

Example

```
EntrySetup: DebugEval(?Symbol = "AAPL" and BarDate = 20240315, RSI(2) < 10)
```

Stops the test and logs evaluation details for AAPL on 2024-03-15.

14.20.148. 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 open position every day of the test, just prior to the **ExitRule** formula evaluation.

If the *DebugExit* formula returns a non-blank string or a non-zero numeric value then whatever it returns is added to the **test debug or log output**.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (see also the debug_and_validate.rts example in Examples-New\12_special_topics):

```
DebugExit: if(debug_sym, Format("c={#2}, c[1]={#2}, held={#}", c, c[1], BarsHeld), "")
```

14.20.149. 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 open position every day of the test, just prior to the **ExitLimit** and **ExitStop** formula evaluations.

If the *DebugTargetStop* formula returns a non-blank string or a non-zero numeric value then whatever it returns is added to the **test debug or log output**.

To make the best use of this debugging feature, use the **IF** and **Format** functions, as in this example (see also the debug_and_validate.rts example in Examples-New\12_special_topics):

```
DebugTargetStop: if(debug_sym, Format("target={#2}, nexthigh={#2}, miss={#2}", long_target, next_high, long_target - next_high), "")
```

14.20.150. DEMA

Category

Multi-Bar Functions

Description

Double Exponential Moving Average

Syntax

DEMA(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

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 / (count + 1).

Double EMA is calculated as *2 * EMA(expr, count) - EMA(EMA(expr, count), count)*.

Unlike the **EMA** function, *DEMA* applies the exponential weight factor from the start of available data rather than beginning with *MA(expr, count)*. This is consistent with how other backtesting software calculates this indicator.

This function supports **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
FastMA: DEMA(C, 21)
```

14.20.151. 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.

Example

```
ExDiv: Dividend > 0
```

Non-zero only on ex-dividend dates.

14.20.152. 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.

14.20.153. DynamicSizing

Category

Strategy Elements

Description

Determines whether strategy operates in a special dynamic sizing mode

Choices

True - use dynamic sizing mode

False - use typical entry/exit formula mode (default)

Notes

If *DynamicSizing* is set to True then most of the typical strategy formulas become disabled and the

Quantity formula determines whether a new position or partial position is entered or exited each day.

In this mode the following elements are disallowed: *EntrySetup, EntryLimit, EntryStop, EntryScore, ExitRule, ExitQty, ExitLimit, ExitLimitQty, ExitStop, ExitStopQty, ExitScore, MaxSameSym, MaxSameCat, MaxSetups, MaxEntries, Side*.

EntryTime and *ExitTime* can be specified but must be the same. The default *NextOpen* makes most sense in this mode, so it is recommended to not specify these times.

Instead of the logic provided by all of the above elements, the backtest engine simply evaluates Quantity once per test period. The value returned is the "target position size" for that period.

If the target size is larger than the current size, an entry is made for the difference.

If the target size is smaller than the current size, an exit is made for the difference.

The sign of the quantity determines the side (long vs. short) of the position to establish.

If the prior position was the opposite side, all current positions are exited and then the new position is entered.

Internally RealTest implements this functionality using multiple positions per symbol, similar to the older way of simulating dynamic sizing.

The key differences are:

- *Quantity* is simply the desired position rather than having to calculate the difference vs. current position
- *EntrySetup* is implicitly True when quantity is larger than current position
- *MaxSameSym* is implicitly infinite
- *ExitRule* is implicitly true when quantity is smaller than current position
- *ExitScore* is implicitly "-Shares" i.e. always exit the smallest sub-positions first
- *Shares* are implicitly *Extern(@thisStrat, Shares)*, i.e., combined quantity of all sub-positions in this strategy for this symbol
- there is no way to access individual sub-positions from strategy formulas though they are still evident in the trade list, log, and orders

DynamicSizing strategies can specify any desired **QtyType** and, as usual, should express their target quantities in that type.

The **Quantity** formula is by far the most important strategy formula in this mode. The **Select** function can be useful to make the typically complex Quantity formula more manageable.

Note that unlike normal strategy mode, where *Quantity: 0* means "no entry", in dynamic mode this means "exit any open position" (go flat).

To specify "no change" in the current position, the Quantity formula should return the constant *nan,* for example:

```
Quantity: Select(C > BBTop(20,2), 100, C < BBBot(20,2), -100, nan)
```

This says "go or stay long 100 when above the upper band, go or stay short 100 when below the lower band, otherwise don't change the position."

Dynamic-sizing strategies can specify any of the usual **capacity constraints** except for those that constrain position counts. Top-down setup selection logic works as usual, including when adding to positions.

See the two example scripts *clenow_stocks_on_move.rts* and *clenow_stocks_on_move_dynamic.rts*. These implement the same strategy logic using (a) the older approach and (b) dynamic sizing mode.

14.20.154. EhlersBP

Category

Multi-Bar Functions

Description

Bandpass filter as described by John Ehlers.

Syntax

EhlersBP(expr, count, bandwidth)

Parameters

expr - data series formula

count - lookback period

bandwidth - passband width

Notes

Isolates a specific frequency band centered on the given period, with the bandwidth parameter controlling how narrow the passband is (smaller values = narrower band).

The count and bandwidth arguments may be formulas rather than constants. Non-constant arguments are re-evaluated on every bar: the filter coefficients are recomputed from that bar's values and the recursion continues from the stored prior outputs, the same adaptive model used by **AEMA**. The formulas may even reference the enclosing Data item's own prior values at offsets of 1 or more, allowing self-tuning filters.

On bars where the period or bandwidth evaluates to n/a, the output is n/a (not 0), and the recursion re-seeds when both are valid again. The effective period is internally clamped to a minimum of 2 bars.

This function supports **one-pass calculation** when used in the **Data Section**.

Example

```
EhlersBP(Close, 20, 0.3)
```

Isolates the frequency band centered on a 20-bar cycle.

Useful for extracting a specific cyclical component from price data while rejecting both trend and noise outside that band.

14.20.155. EhlersHP

Category

Multi-Bar Functions

Description

Two-pole highpass filter as described by John Ehlers.

Syntax

EhlersHP(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

Removes low-frequency components (trends) from the input, passing only cycle activity shorter than the specified period.

Useful for detrending a price series while preserving its cyclical behavior.

The count argument may be a formula rather than a constant. A non-constant period is re-evaluated

on every bar: the filter coefficients are recomputed from that bar's value and the recursion continues from the stored prior outputs, the same adaptive model used by **AEMA**. The period formula may even reference the enclosing Data item's own prior values at offsets of 1 or more, allowing self-tuning filters.

On bars where the period evaluates to n/a, the output is n/a, and the recursion re-seeds when the period next becomes valid. The effective period is internally clamped to a minimum of 2 bars.

This function supports **one-pass calculation** when used in the **Data Section**.

Example

```
EhlersHP(Close, 40)
```

Removes trend components longer than 40 bars from the closing price, leaving only shorter-term cyclical movement.

Useful as a detrending step before cycle analysis.

14.20.156. EhlersSS

Category

Multi-Bar Functions

Description

Two-pole SuperSmoother filter as described by John Ehlers.

Syntax

EhlersSS(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

A low-lag smoothing filter that attenuates frequencies above the cutoff period with less lag than a comparable moving average.

Provides a smoother result than SMA or EMA with sharper frequency rejection.

The count argument may be a formula rather than a constant. A non-constant period is re-evaluated on every bar: the filter coefficients are recomputed from that bar's value and the recursion continues from the stored prior outputs, the same adaptive model used by **AEMA**. The period formula may even reference the enclosing Data item's own prior values at offsets of 1 or more, allowing self-tuning filters.

On bars where the period evaluates to n/a, the output is n/a, and the recursion re-seeds when the period next becomes valid. The effective period is internally clamped to a minimum of 2 bars.

This function supports **one-pass calculation** when used in the **Data Section**.

Example

```
EhlersSS(Close, 10)
```

Smooths the closing price with less lag than a comparable EMA.

Useful as a low-lag trend filter or signal line.

14.20.157. EhlersUS

Category

Multi-Bar Functions

Description

Ultimate Smoother filter as described by John Ehlers.

Syntax

EhlersUS(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

Combines HighPass and SuperSmoother characteristics to remove high-frequency noise while preserving the underlying trend with very little lag.

Produces a smoothed version of the source series on the same scale as the input, preserving the overall trend.

The count argument may be a formula rather than a constant. A non-constant period is re-evaluated on every bar: the filter coefficients are recomputed from that bar's value and the recursion continues from the stored prior outputs, the same adaptive model used by **AEMA**. The period formula may even reference the enclosing Data item's own prior values at offsets of 1 or more, allowing self-tuning filters.

On bars where the period evaluates to n/a, the output is n/a, and the recursion re-seeds when the period next becomes valid. The effective period is internally clamped to a minimum of 2 bars.

This function supports **one-pass calculation** when used in the **Data Section**.

Example

```
EhlersUS(Close, 20)
```

Smooths the closing price over roughly a 20-bar period, removing short-term noise with very little lag while preserving the overall trend.

The result stays on the same scale as price and can be used as a low-lag trend or signal line.

14.20.158. 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 / (count + 1).

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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
EMA50: EMA(C, 50)
```

EMA and XAvg are interchangeable names.

14.20.159. 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

14.20.160. EndDate (Setting)

Category

Settings

Description

The last date to include in a scan or test

Choices

Date Constant - a literal date

Latest - the newest date of any symbol in the current data file

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** (any **StartDate** is then ignored, so a **ScanSettings** section can override dated test dates from **Settings**).

If **DateSym** is specified then *Latest* means the newest date of that specific symbol.

14.20.161. 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.

Example

```
ExitRule: EndOfMonth
```

Returns 1 on the last trading day of each month.

14.20.162. EndOfQuarter

Category

Bar Data Values

Description

Indicates whether this bar is the last bar of a calendar quarter

Notes

Returns 1 for the last bar of a month when month is March, June, September or December, 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 quarter for a specific symbol, use *Month[-1] <> Month and Month % 3 = 0*.

To work correctly when generating tomorrow's orders or using negative (future) offsets, a **HolidayList** should also be provided.

14.20.163. EndOfTestExits

Category

Settings

Description

Controls whether open positions are exited at the end of a backtest

Choices

True - exit all positions at the end of each test (default)

False - do not exit open positions

Notes

Positions exited because the test has ended will show "end of test" in the "Reason" column of the trade list. This setting does not affect **End-of-Data Exits**, which close a position at its symbol's last bar when the data ends before the test does.

When *EndOfTestExits* is False, each position that is still open when the test ends is added to the **trade list** as an "Open Long" or "Open Short" row: the entry columns are filled in, the exit columns show n/a, "Reason" is "open position", and "Bars" is the number of bars held so far. Like skipped setups, these rows are excluded from all trade statistics. Their trade charts (including those saved by **SaveTradeChartsTo**) show the entry marker only, extending to the last bar of the test, so an open position can be reviewed as "currently held" rather than as a trade closed at the end of the test. The rows appear for whichever of *Strategy* and *Benchmark* is selected in **KeepTrades**, and are also written by **SaveTradesAs**.

Possible reasons to disable end-of-test exits are:

- to exclude still-open positions from trade-level stats
- to see and chart the still-open positions as they stand at the end of the test (e.g. when reviewing a live portfolio via **OrdersMode** OrderClerkStats)
- to keep those position records in memory for inspection in the **Debug Panel** (previously this was controlled by the *Debug* choice of the **TestOutput** setting)

14.20.164. 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.

14.20.165. EndOfYear

Category

Bar Data Values

Description

Indicates whether this bar is the last bar of its year

Notes

Returns 1 for the last bar of a year, 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 year for a specific symbol, use *Year[-1] <> Year*.

To work correctly when generating tomorrow's orders or using negative (future) offsets, a **HolidayList** should also be provided.

14.20.166. 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 non-daily BarSize strategy, this will be the actual entry date (e.g. a Monday if Weekly).

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**.

14.20.167. 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 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)
```

14.20.168. EntryPhase

Category

Strategy Elements

Description

Assigns a strategy or benchmark to an entry setup processing phase; lower-numbered phases finalize their orders first

Input

A constant expression (a number or a **Parameter**); the minimum phase is 1

Notes

Each bar, entry setups are collected and finalized (ranked, capped, order flags set) in ascending *EntryPhase* order, and entries fill only after the last phase. No fills or equity effects are visible from one phase to the next, but final order flags and capacity reservations are.

A section in a later phase therefore sees the final order state of every earlier phase - *Extern(@strategy, IsOrder)*, **OrderSum**, **IsSetup** - in both *EntrySetup* and *SetupSkip* formulas. This makes same-bar de-confliction between strategies deterministic. For example, to keep a short strategy out of any symbol that the long strategy is entering or already holds:

```
Strategy: longs
   EntryPhase: 1
   // long entry rules
Strategy: shorts
   EntryPhase: 2
   SetupSkip: Extern(@longs, Shares > 0 or IsOrder)
   // short entry rules
```

Sections that share a phase are finalized together, turn by turn, as described in **Backtest Loop Overview**: declaration order or **StrategyScore** decides which strategy picks first at each turn, and **MaxPerTurn** how many setups it can add per turn. Within a shared phase, another section's same-bar order state is only partially visible: orders granted in earlier turns of the same bar can be seen, but later ones cannot. Use distinct phases whenever complete visibility is needed.

Phase numbers need not be consecutive - only their relative order matters. *EntryPhase* can be inherited from a **Template** via **Using**. It is not valid in a **StatsGroup** or **Combined** section.

Defaults when *EntryPhase* is not specified: each **Benchmark** gets its own phase, in declaration order, before every explicitly numbered phase, and *Strategy* sections without an *EntryPhase* share a single phase after the highest explicit value. This preserves the long-standing default behavior: all benchmarks finalize their orders before any strategy processes its setups.

A benchmark (or strategy) can also be given a **later** phase than other strategies, so that it reacts to their final orders - something the default ordering never allowed.

Phases order setup processing within an entry time. They do not reorder the time-of-day phases of the daily loop - for example, a *ThisClose* section cannot be made to precede a section that enters at the open.

Order generation (Orders mode) processes setups in the same phase sequence, so generated orders always match what the backtest would have done on that date.

EntryPhase was introduced in version 2.0.32.4. In earlier versions, the default ordering described above (benchmarks first, in declaration order, then all strategies together) was the only behavior.

See Also: **SetupSkip**, **IsOrder** and **Backtest Loop Overview**

14.20.169. 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.

14.20.170. 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 specialized formula that is only needed when modeling strategies that allow more setups than entries. It should NOT be used as a general-purpose setup ranking function -- always use **SetupScore** for that purpose. See also **MaxEntries**.

14.20.171. 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
```

14.20.172. 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 "entry skip" 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.

14.20.173. 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 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)
```

14.20.174. 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

formula - a string formula that chooses NextOpen, Intraday or NextClose for each setup -- see Dynamic entry time below

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.

An *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 **STP LMT** with the same qualifiers.

(All of the above order types are stated using Interactive Brokers order nomenclature, and can be automatically generated by adding *OrdersTemplate: Examples\ib_basket_trader.csv* to your **Settings** and running your script in **Orders** mode.)

Dynamic entry time

Instead of one of the choices above, *EntryTime* can be a formula that returns one of the strings *"NextOpen"*, *"Intraday"* or *"NextClose"*, for example to enter less liquid symbols at the close and the rest at the open:

EntryTime: iif(InList("Illiquid"), "NextClose", "NextOpen")

The formula is evaluated for each setup on the same bar as **EntrySetup**, so the choice is already known when **Tomorrow's Orders** are generated, and each order gets the type described above for its time (e.g. **MOC** for a *NextClose* market entry). Setups with different entry times are ranked together and share *MaxPositions* and the other entry limits exactly as they would in a strategy with a single entry time -- only the fill time and order type differ.

The names must be quoted: an unquoted *NextOpen* inside a formula is the next bar's open price. Any other result stops the run with an error, and a result that cannot be calculated (NaN) uses the default time (*NextOpen*, or *Intraday* with an *EntryLimit* or *EntryStop*). *ThisClose* cannot be chosen by a formula, since it is decided a bar later and cannot be ordered in advance -- use a separate strategy for *ThisClose* entries. A strategy specifies either a time name or a formula, not both, and a formula cannot be used with *DynamicSizing*.

When exits can also be *NextClose* (see **ExitTime**) and the strategy has no *ExitLimit* or *ExitStop*, a *NextClose* exit is decided at the open so that the same day's entries can reuse its position slot and capital, as in a strategy that both enters and exits at *NextClose*. With mixed entry times this also lets a *NextOpen* entry use capital that a real account only gets back at that day's close.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

14.20.175. 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.

14.20.176. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
ExpSD: ESD(ROC(C, 1), 20)
```

Exponential standard deviation gives more weight to recent values than StdDev.

14.20.177. 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 or quoted event type name

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.

The type can also be given as a quoted name, e.g. *Event("EarnEst")*, if the event list defined named types. Names are matched without regard to case; a name that does not appear in the event list matches nothing, so the function returns 0.

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) is at or before the bar's time. For daily bars this cutoff is the *ImportBarTime* setting, which defaults to 16:00 (160000).

Example

```
HasEvent: Event(1)
```

Returns the event value if an event of this type exists for this bar, or 0 if none.

14.20.178. 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 **Event List Files** for details on how this mechanism works and can be used.

Events dated after the last imported bar are now retained rather than discarded. This allows formulas such as *Event(n)[-5]* and **UntilEvent**(n) to reference upcoming events (e.g. a known earnings date next week) during order generation, not just during backtests.

14.20.179. ExchangeMap

Category

Settings or **Order 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:

A	B	C	D	E	F	G	H	I	J	K
Exchange	Primary	Market	Match1	Match2	HasOpg	HasMoc	HasLoc	TimeZone	LastMocTime	NearCloseTime
BVME.ETF	BVME.ETF				0	0	0	MET	15:58:00	15:58:00
EBS	EBS				0	0	0	MET	15:58:00	15:58:00
IBIS2	IBIS2		*.MI		0	0	0	MET	15:58:00	15:58:00
AMS	AEB		*.AS		0	0	0	MET	17:38:00	17:38:00
ASX	ASX		*.AX	*.AU	0	0	1	Australia/NSW	15:58:00	15:58:00
GER	IBIS		*.DE	*.BN	0	0	0	MET	16:58:00	16:58:00
PAR	SBF		*.PA		0	0	0	MET	17:28:00	17:28:00
LSE*	LSE		*.L		0	0	0	MET	16:28:00	16:28:00
TSX Venture NEX	NEX		*.CA		0	0	1	US/Eastern	15:48:00	15:58:00
TSX Venture	VENTURE		*.CA		0	0	1	US/Eastern	15:48:00	15:58:00
TSX	TSE		*.CA	*.TO	0	1	1	US/Eastern	15:48:00	15:58:00
CSE	PURE		*.CA		0	0	0	US/Eastern	15:48:00	15:58:00
NEO	AEQLIT		*.CA		0	0	0	US/Eastern	15:48:00	15:58:00
*	AMEX		*		0	1	1	US/Eastern	15:40:00	15:58:00

L	M	N	O	P	Q	R
EarlyLastMocTime	EarlyNearCloseTime	LateLocTime	EarlyLateLocTime	WeeklyLastMocTime	LastOpenTime	AllHours
				15:58:00	9:01:00	0
				15:58:00	10:05:00	0
				15:58:00	9:01:00	0
				17:38:00	9:01:00	0
13:58:00	13:58:00	16:02:00	14:02:00	10:30:00	10:10:00	0
				16:58:00	9:01:00	0
				17:28:00	9:01:00	0
				16:28:00	9:01:00	0
12:48:00	12:58:00			15:48:00	9:31:00	0
12:48:00	12:58:00			15:48:00	9:31:00	0
12:48:00	12:58:00			15:48:00	9:31:00	0
12:48:00	12:58:00			15:48:00	9:31:00	0
12:48:00	12:58:00			15:48:00	9:31:00	0
12:48:00	12:58:00			15:40:00	9:31:00	0

The columns can be in any sequence but must use the names listed below.

The content and purpose of each column is described here:

Column	**Content**	**Purpose**
Exchange	exchange name with optional * or ? **Match** wildcard characters	matches the **?Exchange** of the stock for which an order is being generated
Primary	brokerage exchange	specifies the primary exchange in SMART/primary
Market	symbol matching pattern	allows different symbols on the same exchange to have different settings (e.g. for futures)
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)
HasOpg	0 or 1	1 if this exchange supports open-price-guarantee orders
HasMoc	0 or 1	1 if this exchange supports market-on-close orders else 0
HasLoc	0 or 1	1 if this exchange supports limit-on-close orders else 0

TimeZone	Exchange timezone name in IANA **tz database** 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
EarlyLateLocTime	same as above	ditto on early close days
WeeklyLastMocTime	same as above	Friday "Good Until" time for weekly MOC
LastOpenTime	same as above	GTD time for "at-open" STP orders
FuturesOpenTime	time of day	time after which to increment date when processing fill reports (0 or blank means never do so)
AllHours	0 or 1	always generate GTD rather than DAY orders (set OutsideRTH flag in brokerage order)

The Match1 and Match2 symbol matching patterns are also used to know when to remove the country-specific suffix from a symbol when generating orders, e.g. changing BHP.au to BHP.

If you're generating orders for futures, see also exchangemap_csi.csv or exchangemap_ndu.csv (both are in the Examples folder).

Row Matching Logic

Rows are evaluated top-to-bottom, and the first match is used:

1. *Symbol suffix matching*: The stock's symbol is tested against *Match1* and *Match2* using wildcard matching. If either matches, the row is a candidate.
2. *Exchange matching*: If the stock has a non-empty *?Exchange* field, the row's *Exchange* pattern is matched against it. If the stock has no exchange field, the symbol suffix match alone determines the row.
3. *Market-specific filtering*: If the row has a non-empty *Market* field, the symbol must also match that pattern, or the row is skipped. This allows multiple rows for the same exchange with different session parameters (e.g. different futures products).
4. *Wildcard fallback*: The last row typically has *Exchange* = *, which acts as the default for US equities.

Order matters: more specific rows (e.g. TSX Venture) must appear before less specific ones (e.g. TSX), and the wildcard * row must be last.

Exchange and Order Routing

When RealTest generates an order file, it finds the matching ExchangeMap row for each stock. The *Primary* field is used to construct the exchange routing string *SMART/Primary* which is written into the order's *{exch}* template field.

Example: A stock with exchange ASX maps to *Primary = ASX*, so orders get *SMART/ASX*. A Canadian stock on TSX maps to *Primary = TSE*, producing *SMART/TSE*.

Symbol Suffix Handling

Data providers append suffixes to international symbols (e.g. Norgate uses *BHP.AU*, Yahoo uses *BHP.AX*). IB does not use these suffixes.

When RealTest matches a symbol via *Match1* or *Match2*:

1. If the match was on a country suffix pattern like *.AU, the dot and suffix are removed (e.g. *BHP.AU* becomes *BHP*).
2. If the dot is a symbol class separator (e.g. *BRK.B*) and the match pattern had no dot, the dot is by default replaced with a space for IB format (*BRK B*). The **OrdersClassDelim** setting can change this conversion for order lists aimed at other brokers.

Order Type Selection

Three boolean flags control order type availability per exchange:

1. *HasMoc*: If true, exit orders for "next close" can use the MOC order type. If false, a timed MKT order with a good-after-time is used instead.
2. *HasLoc*: If true, exit limit orders for "next close" can use the LOC order type. If false, a timed LMT order is used instead.
3. *HasOpg*: If true, market-on-open entry orders use OPG TIF. If false, DAY (or GTC for AllHours markets) TIF is used.

Order Timing

When generating orders, RealTest combines the time value from the matched ExchangeMap row with the timezone to produce a full timestamp (e.g. *15:40:00 US/Eastern*).

On early-close days (as defined in the HolidayList), the "Early" variants of each time field are used instead:

1. *LastMocTime* → *EarlyLastMocTime*
2. *NearCloseTime* → *EarlyNearCloseTime*
3. *LateLocTime* → *EarlyLateLocTime*

For weekly (or larger) bar strategies, *WeeklyLastMocTime* is used instead of *LastMocTime*.

All-Hours Trading

When *AllHours = 1*:

1. DAY TIF is never used. Entry orders use GTC instead. Exit orders use GTD.
2. The *{allhours}* template field outputs 1, which can set the OutsideRTH flag on IB orders.
3. Entry limit orders on non-daily strategies use GTD with *NearCloseTime* expiry.

Futures Trade Import Date Adjustment

The *FuturesOpenTime* column is used during trade import. Futures like CME products have sessions that begin in the evening (e.g. 5:00 PM) and run through the next calendar day. A trade executed at 6:00 PM on Monday belongs to Tuesday's daily bar.

RealTest checks each imported trade's execution time against *FuturesOpenTime*. If the trade time is at or after this threshold, the trade date is incremented by one trading day.

14.20.180. ExcludeIf

Category

Import Specification

Description

Import filter formula (exclude symbol if true)

Input

Any formula specifying a true/false condition (non-zero means true)

14.20.181. 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.

14.20.182. 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 evaluated using **the day prior to exit day** as the most recent bar in the formula, except when **ExitLimitTime** is *ThisClose*, which uses the exit-day bar.

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**.

14.20.183. 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.

14.20.184. 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* is 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.

An *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**.

14.20.185. 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.

14.20.186. 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
```

14.20.187. 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.

14.20.188. 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.

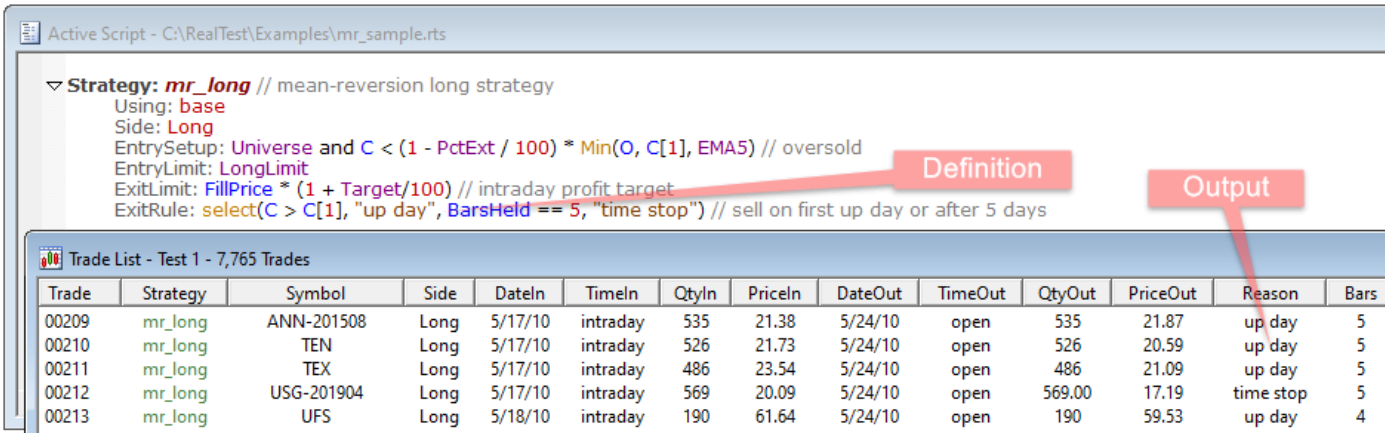
When exit time is *NextOpen* (the default) or *NextClose*, *ExitRule* is evaluated using **the day prior to exit day** as the most recent bar in the formula. For *ThisClose* exits, the exit day is the most recent bar.

This formula is typically used to specify 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:



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 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**.

14.20.189. 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.

14.20.190. 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 by default 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 evaluated using **the day prior to exit day** as the most recent bar in the formula, except when **ExitStopTime** is *ThisClose*, which uses the exit-day bar.

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**.

14.20.191. 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.

14.20.192. 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* is 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.

An *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**.

14.20.193. 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)

Intraday - interpreted as NextOpen (a plain ExitRule has no intraday trigger; use ExitStopTime or ExitLimitTime for intraday stop or limit exits)

NextClose - exits occur at tomorrow's close

formula - a string formula that chooses NextOpen or NextClose for each position -- see Dynamic exit time below

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.

Dynamic exit time

Instead of one of the choices above, *ExitTime* can be a formula that returns the string *"NextOpen"* or *"NextClose"* (*"Intraday"* means *NextOpen*), for example to exit at Friday's close rather than Friday's open when the exit is decided on a Thursday:

ExitTime: iif(DayOfWeek = 4, "NextClose", "NextOpen")

The formula is evaluated for each position on the same bar as **ExitRule** (Thursday's bar in the example), so the choice is already known when **Tomorrow's Orders** are generated and each exit order gets the matching type (**MKT** on the open or **MOC**). It applies only to *ExitRule* exits;

ExitLimitTime and *ExitStopTime* are unchanged.

The names must be quoted: an unquoted *NextOpen* inside a formula is the next bar's open price. Any other result stops the run with an error, and a result that cannot be calculated (NaN) means *NextOpen*. *ThisClose* cannot be chosen by a formula. A strategy specifies either a time name or a formula, not both, and a formula cannot be used with *DynamicSizing*. See **EntryTime** for how a *NextClose* exit's capital is reused by same-day entries.

For more information on how the backtest engine works in general, see **Backtest Engine Details**.

14.20.194. 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.

14.20.195. ExitType

Category

Current Position Information

Description

Returns the exit order type for which an OrderExtra formula is being evaluated

Notes

The return value is the "exit reason" as displayed in a trade list (string) or accessed by **T.Reason** (numeric).

ExitType can be used in an **OrderExtra** formula to enable different output for each type of exit order being generated.

For example if a custom **OrdersTemplate** specifies *extra1* rather than *gtd* in row 2 under the *GoodAfterTime* header, it could specify different activation times for the target and the stop orders:

```
OrderExtra1: Format("{?} {?} US/Eastern", Format(?OrderDate, "yyyy-MM-dd"),
if(ExitType = "exit limit", "10:00:00", "11:00:00"))
```

14.20.196. Exp

Category

General-Purpose Functions

Description

Exponential function (e^x)

Syntax

Exp(value)

Parameters

value - formula

Example

```
Growth: Exp(Sum(Log(1 + ROC(C, 1) / 100), 20))
```

Compound return via log-sum-exp, equivalent to Product.

14.20.197. 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 and position information.

Extern allows you to 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)*

If a stock symbol already starts with a $, e.g. $SPX, you will have to add another $, hence *Extern($$SPX, C)*.

The special symbol *$CalSym* can be used to reference the **CalendarSym** of the current strategy.

The special symbol *$Global* references a synthetic symbol that has a bar on every date in the loaded data (with no price data). This is useful for date-based formulas that need a value on every date, independent of any particular symbol.

See also **Symbol References**.

There are also special types of *Extern* references available for **individual futures contracts** and **corresponding industry index** symbols.

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 reference the automatically-added **Tracker** benchmark of the current strategy, use @Tracker.

When a @strategy or @group reference reads same-bar order state (**IsOrder**, **OrderSum**, etc.) during entry setup processing, the values are final only if the referenced section is in an earlier **EntryPhase**; within a shared phase they reflect only the selection turns completed so far.

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.

It is also possible to refer to a dynamic (non-constant) external strategy or symbol, either by string or number, using **StratRef** or **SymRef**.

The Extern function can be nested, e.g. *Extern(@mystrat, Extern($XYZ, BarsHeld))* to determine how long *mystrat* has held a position in *XYZ.*

Examples

`SpyClose: Extern($$SPX, C)`

Reference another symbol's data. Use $ prefix; for symbols starting with $, use $$.

`StratEquity: Extern(@MyStrategy, S.Equity)`

Reference another strategy's stats using the @ prefix.

14.20.198. Extra

Category

Bar Data Values

Description

Unadjusted close when data is kept adjusted, or turnover value from Norgate import, or user-defined bar value from CSV import

Notes

When an import keeps prices split- and dividend-adjusted (**KeepAdjusted**), the *Extra* field instead holds each bar's unadjusted (as-traded) close, which is otherwise unavailable from adjusted data.

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.

14.20.199. 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:

Active Script - C:\RealTest\Scripts\Examples\fundamentals.rts

```
// 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
```

items to import

imported item values

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

14.20.200. FeesInOut

Category

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 **FeesList** rather than *FeesInOut*.

Unlike **CashInOut**, *FeesInOut* is included in all the stats calculations of a backtest, such as the

annual return, max drawdown, and daily net percent gain or loss.

Examples

Pay $120/month to your broker for realtime data services:

```
FeesInOut: if(EndOfMonth, -120, 0)
```

See also the **management_fees.rts** example script.

14.20.201. FeesList

Category

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 **FeesInOut** rather than *FeesList*.

Unlike **CashList**, items in *FeesList* are included in all the stats calculations of a backtest, such as the annual return, max drawdown, and daily net percent gain or loss.

14.20.202. 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.

The *FillFraction* of each trade becomes the **T.Fraction** value in the trade list.

14.20.203. 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.

14.20.204. FillPriceAvg

Category

Current Position Information

Description

Average entry fill price of the current symbol when pyramiding (**MaxSameSym** > 1)

Notes

FillPriceAvg returns the share-weighted average entry fill price of all open positions of the current symbol - for example the legs of a pyramided position (**MaxSameSym** > 1), or the positions held by several strategies in a group.

FillPriceAvg returns nan when referenced directly for the current strategy's own position, so it must be wrapped in **Combined** (the current strategy and any strategies grouped with it) or **Extern** (another strategy).

14.20.205. FillPriceMax

Category

Current Position Information

Description

Highest entry fill price of the current symbol when pyramiding (**MaxSameSym** > 1)

Notes

FillPriceMax returns the highest entry fill price of all open positions of the current symbol - for example the legs of a pyramided position (**MaxSameSym** > 1), or the positions held by several strategies in a group.

FillPriceMax returns nan when referenced directly for the current strategy's own position, so it must be wrapped in **Combined** (the current strategy and any strategies grouped with it) or **Extern** (another strategy).

14.20.206. FillPriceMin

Category

Current Position Information

Description

Lowest entry fill price of the current symbol when pyramiding (**MaxSameSym** > 1)

Notes

FillPriceMin returns the lowest entry fill price of all open positions of the current symbol - for example the legs of a pyramided position (**MaxSameSym** > 1), or the positions held by several strategies in a group.

FillPriceMin returns nan when referenced directly for the current strategy's own position, so it must be wrapped in **Combined** (the current strategy and any strategies grouped with it) or **Extern** (another strategy).

14.20.207. 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
```

14.20.208. 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**.

14.20.209. 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-Filter Scan** for details and an example.

14.20.210. Find

Category

String Functions

Description

Find a string within another string and return its offset

Syntax

Find(string, substring)

Parameters

string - a **literal string** enclosed in either double or single quotes ("string" or 'string') or a **string function**

substring - a string (or function) containing the substring to find

Return Value

The offset to the substring if found, starting with 1 if found at the beginning of the string, or 0 if not found.

Notes

All string comparisons in RealTest are case-insensitive.

Example

```
HasDash: Find(?Symbol, "-")
```

Returns the position (starting at 1) if found, or 0 if not found.

14.20.211. 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 of 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.

Example

```
ScanCol: Format("RSI={%1} ATR=${%2}", RSI(14), ATR(14))
```

Use {%N} for numeric and {?} for string format specifiers.

14.20.212. FormatDate

Category

String Functions

Description

Convert a date to a string in any format

Syntax

FormatDate(date, format)

Parameters

date - numeric date value such as **BarDate**, 20240831, etc.

format - **Date Format Picture**

Notes

In script-specified output such as **Scan** items with {//} as their **output format**, dates are automatically formatted using the currently defined **DateDisplay** or the default as defined in **Program Options**.

FormatDate is provided for the rare case where you need a specific date formatted differently from the current setting.

Example

```
DateStr: FormatDate(BarDate, "MM/dd/yyyy")
```

Formats a numeric date using Windows date format picture strings.

14.20.213. Fundamentals

Category

Import Specification

Description

List of one or more **current** fundamental items to include when importing **Norgate** data

Notes

Norgate provides more than 150 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:

Active Script - C:\RealTest\Scripts\Examples\fundamentals.rts

```
// 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
```

items to import

imported item values

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

14.20.214. FunBar

Category

Bar Data Values

Description

Ordinal number of a data bar within the calculation of a multi-bar function.

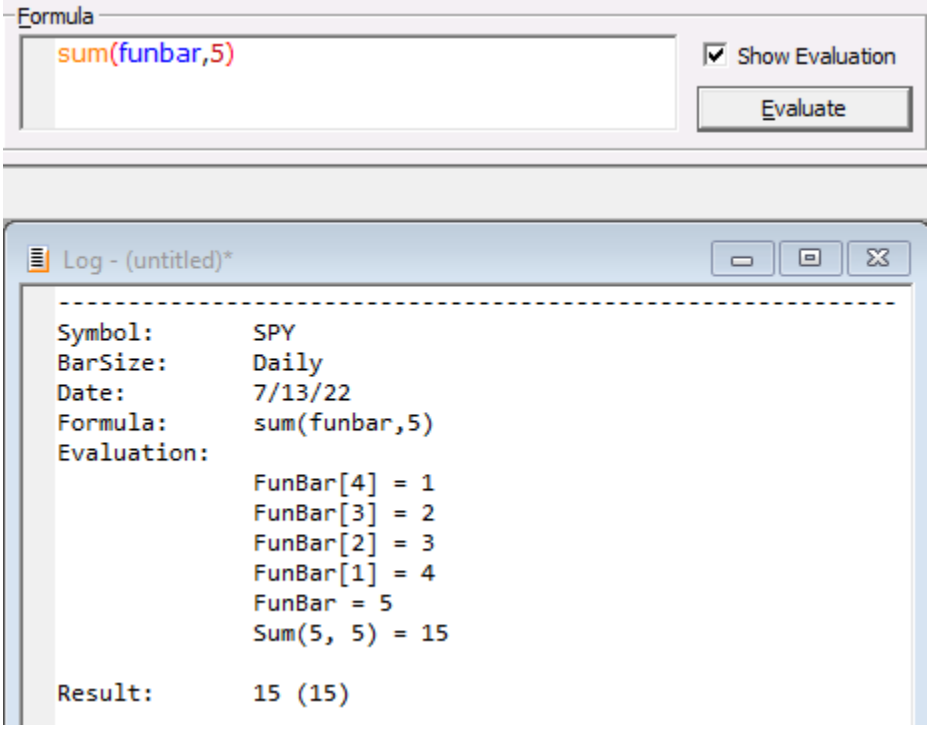
Notes

A typical use case for *FunBar* is as the second argument to the **Correl** function when correlating a data series with a linear growth line.

See also **FunStat** for the stats series version.

FunBar makes it possible, in some cases, to calculate with a one-line function call a complex formula that otherwise needs 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**.

Example

```
WeightedMA: Sum(C * FunBar, 10) / Sum(FunBar, 10)
```

FunBar counts from 1 (oldest) to count (newest) within the function's lookback window.

14.20.215. FunStat

Category

Test Statistics Arrays

Description

Ordinal number of a stat period within the calculation of a multi-bar function

Notes

A typical use case for *FunStat* is as the second argument to the **Correl** function for correlating a stats series with a linear growth line.

See also **FunBar** for the data series version.

14.20.216. FxRatio

Category

Current Position Information or **Bar Data Values**

Description

Foreign currency exchange ratio for this position on this date

Notes

If **Currency** was specified and the current symbol does not trade in that base currency and the needed FX conversion symbol was also imported then *FxRatio* will report the current currency conversion ratio for this position on this date.

If any of the above conditions is not true then *FxRatio* will simply return 1.

Since RealTest manages currency conversions at position entry and exit and adjusts daily mark-to-market position values by FX ratio, it's usually not necessary to use this ratio variable in your strategy formulas.

One case where FxRatio may be useful is in a **Quantity** formula when **QtyType** is *Shares*. That quantity type is interpreted verbatim. *Quantity: 10* means buy (or sell) 10 shares (or contracts), regardless of the instrument's currency. If you want to calculate a share quantity and then FX-adjust it then divide it by FxRatio.

14.20.217. GetField

Category

String Functions

Description

Extract a delimited substring from a string

Syntax

GetField(string, delimiter, instance)

Parameters

string - a **literal string** enclosed in either double or single quotes ("string" or 'string') or a **string function**

delimiter - a string (or function) containing the delimiter used to mark the start and/or end of a field

instance - the instance number of the field to return (1 being the first)

Return Value

The extracted field as a string.

Notes

Though *delimiter* is typically one character such as "," it can also be multiple characters.

Example

```
GetField("a-b-c", "-", 2)
```

returns "b".

14.20.218. GreedyScoring

Category

Strategy Elements

Description

Determines whether strategy uses a special variant of top-down setup selection

Choices

True - use greedy scoring mode

False - don't use greedy scoring mode (default)

Notes

GreedyScoring mode provides a way to employ a **greedy algorithm** within the top-down setup selection process.

When enabled, this mode causes **SetupScore** to be re-evaluated for each setup at each step of the selection loop.

This allows a kind of dynamic scoring process that includes each already-selected setup in the score for the next one.

The main use case for this mode is to produce an optimally uncorrelated portfolio.

The algorithm to do so works like this:

> *while setup slots remain, select the available setup with the lowest average correlation to open positions and other setups selected so far*

Specifically in a RealTest script (from the *optimal_diversity.rts* example):

> *SetupScore: Select(OrderSum(1) = 0 and PositionSum(1) = 0, score1, -CorrelAvg(IsOrder or Shares > 0, roc1, bars))*

In this example "score1" is calculated in Data and is only used when there are no open positions and no other setups selected yet, so nothing to correlate the new setup with:

> *score1: if(setup, -CorrelAvg(setup, roc1, bars), nan)*

As can easily be imagined, re-evaluating **CorrelAvg** for a large number of symbols at each step of each period's setup selection loop can become VERY time consuming.

This example restricts its universe to the S&P 100 (OEX) and operates on a weekly timeframe. Those constraints make backtest performance tolerably fast.

There may be other use cases for *GreedyScoring* mode that do not involve correlation calculations.

14.20.219. Graphs

Category

Script Sections

Description

Daily stats graph content definitions

Notes

See **Graphs Section** and **Daily Stats Graph Windows**.

14.20.220. 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.

14.20.221. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
High52: Highest(H, 252)
```

Highest and HHV are interchangeable names.

14.20.222. 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)).*

Example

```
HullMA: HMA(C, 20)
```

HMA and HAvg are interchangeable names.

14.20.223. HolidayList

Category

Settings or **Strategy Elements** or **Order Settings**

Description

Specifies the path to a text file which contains a list of future holidays, or "us_auto", "au_auto" or "ca_auto"

Notes

The purpose of the holiday list is to enable RealTest to know the next market date when generating orders.

For backtesting, RealTest does not require a holiday list because there are simply no data bars with market holiday dates.

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 if both are provided.

Instead of a file name, you can specify one of the following and RealTest will calculate the standard market holidays and early-close days for the next five years, starting with the year of the latest date in the data:

HolidayList: us_auto - US stocks, using NYSE holiday rules

HolidayList: au_auto - Australian stocks, using ASX holiday rules (including the 2:10pm early closes on Christmas Eve and New Year's Eve)

HolidayList: ca_auto - Canadian stocks, using TSX holiday rules (including the 1:00pm early close on Christmas Eve)

Add "_log" to any of these, e.g. *HolidayList: au_auto_log*, to cause the generated holiday list to be written to a log window when a test is run.

The calculated lists follow each exchange's standard rules and do not include one-time closures such as the NYSE closure on 2025-01-09. If an exchange announces such a closure, use a holiday list file instead.

To specify holidays for any other market, provide a text file with one holiday date per line.

The *Examples* folder includes holiday list files for US, AU, and CA.

Only future market holidays need to be listed (typically there are about 10 per year).

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).

14.20.224. Hurst

Category

Multi-Bar Functions

Description

Hurst exponent (trending vs mean-reverting)

Syntax

Hurst(expr, count, maxlag {20})

Parameters

expr - data series formula

count - lookback period (minimum 8)

maxlag - largest lag used in the estimate -- optional, default 20 (clamped to count / 2)

Notes

Estimates the Hurst exponent of the series over the lookback window using the variance-of-lagged-differences method: the standard deviation of the lag-t differences is computed for each lag from 2 to maxlag, and H is the slope of the least-squares regression of log(std dev) on log(lag).

A value near 0.5 indicates a random walk (no memory). Values above 0.5 indicate persistence (a series that trends), and values below 0.5 indicate anti-persistence (a series that mean-reverts).

The typical use is as a regime filter: favor trend-following entries while H > 0.5 and mean-reversion entries while H < 0.5.

Use generous lookbacks -- 100 bars or more. The estimate is statistically noisy in short windows, and since the output is the raw regression slope (not clamped to the 0-1 range), short-window readings can even be negative. Read it as a slowly-changing regime indication rather than a bar-to-bar signal.

Example

```
Trending: Hurst(Close, 252) > 0.5
```

True when the symbol has been in a persistent (trending) regime over the past year of daily bars.

14.20.225. HVOL

Category

Indicator Functions

Description

Historical volatility

Syntax

HVOL(len, ppy {S.BPY})

Parameters

len - lookback period

ppy - number of periods per year (optional)

Notes

HVOL(len) could be calculated as *StdDev(log(c/c[1]),len) * 100 * Sqr(ppy).*

ppy defaults to **S.BPY** if the current bar size is Daily, 52 for Weekly, 12 for Monthly, 4 for Quarterly, 1 for Yearly.

This function supports **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
HV20: HVOL(20)
HV20w: HVOL(20, 52)
```

The second line specifies 52 periods per year for weekly bar data.

14.20.226. ICDF

Category

General-Purpose Functions

Description

Inverse Cumulative Distribution Function

Syntax

ICDF(value)

Parameters

value - formula

Notes

The inverse **CDF** converts cumulative probabilities into data values for a specified distribution, such as the normal distribution. This aids in trading indicator development by mapping normalized data back to meaningful price levels, helping to identify significant market thresholds.

Example

```
Threshold: ICDF(0.95)
```

The z-score value at the 95th percentile of a normal distribution (about 1.645).

14.20.227. 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 of 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.

Example

```
Dir: if(C > MA(C, 200), 1, -1)
```

Returns 1 in an uptrend, -1 in a downtrend.

14.20.228. IgnoreDividends

Category

Strategy Elements

Description

Determines whether trade profits in a test include dividends paid while the position was held

Choices

True - don't include dividends in trade and test P/L

False - include dividends in trade and test P/L (default)

Notes

This strategy-level setting is only relevant if your imported data includes dividends as per-share amounts paid on ex-dividend dates rather than as dividend-adjusted prices.

The default **Adjustment** mode for most **DataSource** choices (e.g. Norgate) is *Capital*, i.e. only adjust for stock splits, not for dividends.

In this mode, RealTest by default applies each dividend received to the P/L of the trade and the test.

For a long position, the dividend is a credit. For a short position it is a debit.

Most users prefer to see the impact of dividends in their test results without having to use total-return adjustment.

If you need to see results without dividends, e.g. for comparison with other software that doesn't support direct application of dividends to P/L, this strategy-level setting can be used to exclude dividends from your results.

Dividends that are included in P/L accrue to the open position on the ex-date and become free cash when the position exits; see the **DividendsToCash** setting to count them as free cash from the ex-date instead.

14.20.229. Import

Category

Script Sections

Description

Data import specification

Notes

See **Import Section**.

14.20.230. 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**.

See also **Include Section**.

14.20.231. 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 (from the first DataSource that includes it).

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 alias syntax is also the way to keep trading a stock that the data vendor has temporarily renamed while an open position is held under the old symbol, e.g. *HGODB.au>HGO.au* during an ASX consolidation; see **Symbol Changes in Open Positions**.

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 returned 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 *nth 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"//}.

14.20.232. 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.

Example

```
InIndex: InSPX
```

Returns 1 if the stock is a current constituent. Replace SPX with any imported index symbol.

14.20.233. 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.

14.20.234. InfoDelist / InfoExpiry

Category

Stock/Contract Information

Description

Stock delisting date or futures contract expiration date

Notes

When using **CSI Futures** continuous contract data, *InfoExpiry* returns the expiration date of the individual contract that would be held on the date of the **current bar** for which the formula is being evaluated.

When using **Norgate** futures data, InfoExpiry behaves the same way for continuous series, returning the expiration date of the underlying contract held on the current bar; for an individual contract symbol it returns that contract's own expiration date.

With *Norgate* stock data, InfoDelist shows the delisting date for delisted symbols.

To provide this field in your own **SymInfoFile**, use the *DelistDate* column.

Example

```
NearExpiry: Days(BarDate, InfoExpiry) < 30
```

Days until contract expiration or stock delisting date.

14.20.235. 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.

14.20.236. 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.

14.20.237. 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 *Margin* column.

Note that this is just a current value, not a historical data series.

14.20.238. 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**.

To provide this field in your own **SymInfoFile**, use the *Shares* column.

This is not a historical data series. The number returned is always the current value.

14.20.239. InfoTRBC

Category

Stock/Contract Information

Description

The Refinitiv 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.

14.20.240. 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 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.

14.20.241. InSet

Category

General-Purpose Functions

Description

Whether a value is in a specific set of values

Syntax

InSet(value, value1, value2, value3, ...)

Parameters

value - the value to look up

value1...valueN - values to compare it to

Return Value

1 (true) if value is one of the set of values, 0 (false) otherwise

Notes

InSet(Symbol, $AAPL, $GOOGL, $MSFT)

is equivalent to:

Symbol = $AAPL or Symbol = $GOOGL or Symbol = $MSFT

Arguments can be numeric or strings. If strings then comparison is case-insensitive.

Example

```
IsTech: InSet(?Sector, "Technology", "Communication Services")
```

Returns 1 if the stock's sector matches any value in the set.

14.20.242. 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.

These 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)
```

When IsExit is used in a boolean expression such as in the above example, it is interpreted as a number (0 means false, 1 means true in all boolean conditions in RealTest).

IsExit itself is always numeric (0 or 1). To obtain the exit reason code as a string - for example to store it in an **OrderExtra** variable - use **ExitType**, which returns the exit reason string when the current formula pertains to an exit, as shown below.

```
OrderExtra3: Format("{?}", if(IsExit, ExitType, ""))
```

This stores the exit reason string in the *extra3* variable for **OrdersTemplate** output.

14.20.243. 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.

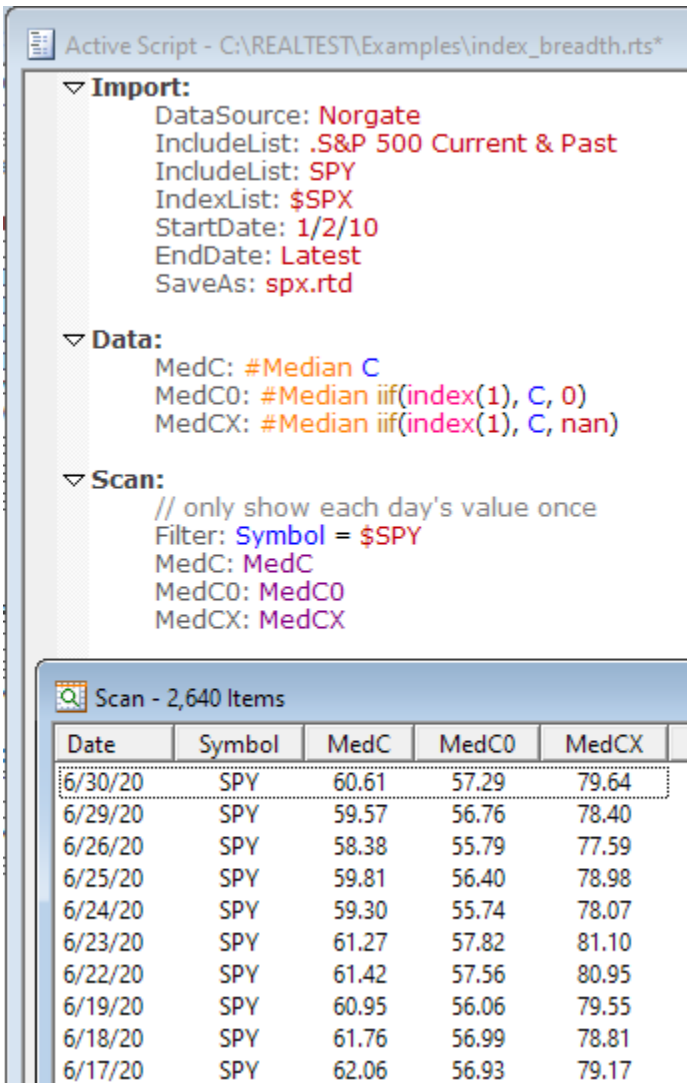
Once any term of any formula evaluates to NAN, the result of the entire formula becomes NAN. This is why it can be useful to find out if a specific term is NAN before evaluating the entire formula. The **IIF** function is useful for this purpose.

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**:



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).

MedC0 filters out non-constituents, but by substituting 0 for their prices, it skews the median value even lower than 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.

Example

```
HasData: 1 - IsNaN(C)
```

Returns 1 if close data is available for this bar, 0 if not.

14.20.244. IsOrder

Category

Current Position Information

Description

Returns 1 (true) if the current stock passed **EntrySetup** for the specified strategy and became an actual entry order - that is, it was not skipped due to a position or allocation limit (**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".

When another strategy's *IsOrder* is referenced during same-bar setup processing (in *EntrySetup* or *SetupSkip*), the value is final for strategies in an earlier **EntryPhase**. Within the same phase, it reflects only the orders granted in earlier selection turns of the current bar.

14.20.245. 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.

When another strategy's *IsSetup* is referenced during same-bar setup processing, the value is final for strategies in an earlier **EntryPhase**; within the same phase it reflects only the processing completed so far in the current bar.

14.20.246. IsTracker

Category

Current Strategy Information

Description

Indicates whether the strategy is being processed as itself or as its tracking benchmark

Notes

When a strategy specifies **Tracker** as either *Visible* or *Hidden*, a **Benchmark** copy of the strategy is internally added.

This means that each formula defined by the strategy is evaluated twice per test day: once for the strategy and once for its tracking benchmark.

IsTracker returns 1 (true) while the benchmark copy is being evaluated and 0 (false) while the strategy itself is being evaluated, so it can be used in any formula that needs to behave differently depending on which one is being evaluated.

See the **mr_sample_tracking.rts** example script for a simple example of *Tracker* and *IsTracker* usage.

See the **oex_tf_auto_track.rts** example script and its description in the **StratData** section topic for a more complex example.

14.20.247. 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.)

Example

```
DynRef: Item(Format("MA{}", Param1))
```

Dynamically reference a Data item whose name includes a parameter value.

14.20.248. JDSAR

Category

Indicator Functions

Description

Directional variant of the Parabolic SAR

Syntax

JDSAR(dir {0}, accel {0.02}, max {0.2}, len {100}, jdmode {0})

Parameters

dir {0} - which stop to return: > 0 = long-side SAR (trails below price), < 0 = short-side SAR (trails above price), 0 = whichever side is currently active

accel {0.02} - initial acceleration factor

max {0.2} - maximum acceleration factor

len {100} - number of bars to use in the calculation

jdmode {0} - 0 = standard high/low trigger; non-zero = close-based variation (the stop reverses only when the close breaks it, ignoring intrabar penetration)

Notes

Unlike the standard **SAR**, which maintains a single stop-and-reverse line, *JDSAR* computes a long-side and a short-side parabolic stop independently on every bar. The *dir* argument selects which one to return.

This makes it possible to attach a parabolic trailing stop to a specific side of a position that was entered by some other signal, independent of the indicator's own trend state. For a long position, *JDSAR(1)* always returns the long-side (below-price) stop; for a short position, *JDSAR(-1)* returns the short-side (above-price) stop.

With *dir* = 0 (the default), *JDSAR* returns whichever side is currently active, behaving like a conventional flipping SAR.

When *jdmode* is non-zero, the stop is triggered by each bar's close rather than its high or low, so intrabar wicks do not cause a reversal.

Examples

`ExitStop: JDSAR(1)`

Long-side parabolic trailing stop, default parameters.

`ExitStop: JDSAR(-1, 0.02, 0.2, 100, 1)`

Short-side stop using the close-based (jdmode) variation.

14.20.249. 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** for further details.

This function supports **one-pass calculation** when used in the **Data Section** with non-variable length parameters.

Example

`AdaptMA: KAMA(C, 10, 2, 30)`

The most commonly used Kaufman Adaptive Moving Average parameters.

14.20.250. KBBot

Category

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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Example

```
KeltBot: KBBot(20, 1.5)
```

14.20.251. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Example

```
KeltTop: KBTop(20, 1.5)
```

14.20.252. 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.

If *True* is selected and the data source provides it, the **Extra** field is used to store the unadjusted

close in each bar.

See **Split Handling** for details on how RealTest uses unadjusted data with knowledge of split factors to provide realistic as-traded backtests.

14.20.253. KeepNonIncluded

Category

Import Specification

Description

Whether to keep all files found in local file data path when **IncludeList** is used

Choices

False - only import the symbols in the *IncludeList* if specified (default)

True - import all files (symbols) found even when IncludeList is specified

Notes

With database (whether local or cloud) data sources (Norgate, Tiingo, Yahoo, etc.) it is always true that only the symbols listed in one or more *IncludeList* definitions will be imported.

With local file data sources (CSV, CSIFutures, CSIStocks, MetaStock) we often want to simply import all the files found in the specified **DataPath** locations.

By default, with local file sources, the use of an *IncludeList* acts like a filter, meaning "only include these symbols".

KeepNonIncluded enables *IncludeList* to be used to define named/numbered subsets of the available symbols while still importing all of them.

14.20.254. 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 ", " series ", or " n-vtg " (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:

removed         GOOG            Alphabet Inc Class C Common
kept            GOOGL           Alphabet Inc Class A Common

removed         BATRA           Atlanta Braves Holdings Inc Series A Common
kept            BATRK           Atlanta Braves Holdings Inc Series C Common

removed         CHTRQ-200911    Charter Communications Inc Class A Common
kept            CHTR            Charter Communications Inc Class A Common

removed         CMCSK-201512    Comcast Corp Class A Special Non-Voting Common
kept            CMCSA           Comcast Corp Class A Common

removed         FOX             Fox Corp Class B Common
kept            FOXA            Fox Corp Class A Common

removed         LBTYK           Liberty Global Ltd Class C Common
kept            LBTYA           Liberty Global Ltd Class A Common

removed         LILA            Liberty Latin America Ltd Class A Common
kept            LILAK           Liberty Latin America Ltd Class C Common

removed         LMCK-201604     Liberty Media Corp Series C Common
kept            LMCA-201604     Liberty Media Corp Series A Common

removed         TECU-201509     Tecumseh Products Co Common
kept            TECUA-201405    Tecumseh Products Co Class A Common

removed         TFCF-201903     Twenty-First Century Fox Inc Class B Common
kept            TFCFA-201903    Twenty-First Century Fox Inc Class A Common
```

14.20.255. 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 be used.

Strategy and *Benchmark* select whose trades are kept; *Skipped* additionally keeps the skipped setups of those same strategies (so *Skipped* on its own keeps nothing). Positions left open by **EndOfTestExits**: False are listed as "Open" rows following the same *Strategy*/*Benchmark* selection.

14.20.256. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **StdDev** and **Skewness**.

Example

```
Kurt: Kurtosis(ROC(C, 1), 100)
```

14.20.257. 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

Example

```
Prefix: Left(?Symbol, 3)
```

First 3 characters of the symbol name.

14.20.258. 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

Example

```
SymLen: Length(?Symbol)
```

14.20.259. Library

Category

Script Sections

Description

Named formulas calculated when they are referenced, using the current context

Notes

See **Library Section** for a more detailed description.

14.20.260. 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**.

14.20.261. 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**.

14.20.262. 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 0 to *count*-1 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.

Example

```
LR20: LinReg(C, 20)
```

Endpoint of 20-bar linear regression of closing prices.

14.20.263. 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 **cii_rotate.rts** in the *Examples* folder for further clarification on how this CII mechanism works.

14.20.264. 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 *Log(S.Equity/S.Equity[1]).*

Weekly would be *Log(S.Equity/S.Equity[5]).*

The log return of a trade (for display in the trade list window, assuming long side) would be *Log(T.PriceOut/T.PriceIn).*

Example

```
LogRet: Log(C / C[1])
```

Daily log return.

14.20.265. LogDetail

Category

Settings

Description

Specifies which categories of output to include in the **Test Details Log**

Choices (multiple, separated by commas)

All - include every type of log output (the default, and equivalent to listing all the choices below)

Caps - list the strategy's active capacity limits (MaxSetups, MaxEntries, MaxPositions, exposure/investment caps, etc.) for the bar

Cashflow - show daily cash, margin, interest, dividend and currency-conversion lines

Positions - show the snapshot of open positions written at the end of each bar

Setups - show the per-bar list of strategy setups (candidate entries ranked and assessed)

Trades - show entry and exit transactions

Notes

This setting only affects the **Test Details Log**, which is only generated when **TestOutput** includes *Log*.

Omitting one or more categories produces a smaller log file focused on just the parts you want to examine. For example, *LogDetail: Trades, Positions* suppresses the per-bar setup-ranking and cashflow lines.

The *Setups* section is always suppressed for bars on which no setups exist, even when *Setups* is included.

Example

```
LogDetail: Trades, Setups
```

Restricts the test details log to setup-ranking output and entry/exit transactions.

14.20.266. 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.

Active Script - C:\REALTEST\temp.rts

```
Import:
    DataSource: Norgate
    IncludeList: .Dow Jones Industrial Average
    ExcludeIf: C > 100
    StartDate: 1/2/20
    EndDate: Latest
    SaveAs: dji.rtd
    LogFile: dji.log
```

Log - C:\RealTest\dji.log

```
-----------------------------------------------------------
Importing Norgate Data From NDU
-----------------------------------------------------------
Symbol          FirstDate       LastDate        Bars            Events
AAPL            was filtered
BA              was filtered
DIS             was filtered
IBM             was filtered
CSCO            1/2/20          6/30/20         125             1
GS              was filtered
JNJ             was filtered
KO              1/2/20          6/30/20         125             2
MMM             was filtered
MSFT            was filtered
PFE             1/2/20          6/30/20         125             2
RTX             1/2/20          6/30/20         125             2
UNH             was filtered
WMT             was filtered
VZ              1/2/20          6/30/20         125             2
CAT             was filtered
AXP             1/2/20          6/30/20         125             1
DOW             1/2/20          6/30/20         125             2
CVX             1/2/20          6/30/20         125             2
INTC            1/2/20          6/30/20         125             2
HD              was filtered
MCD             was filtered
JPM             1/2/20          6/30/20         125             2
NKE             1/2/20          6/30/20         125             2
TRV             was filtered
MRK             1/2/20          6/30/20         125             2
PG              was filtered
WBA             1/2/20          6/30/20         125             2
V               was filtered
XOM             1/2/20          6/30/20         125             2

Saved As C:\RealTest\dji.rtd

Import Was Completed
```

The "Dividends" 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 is shown in the "EventList" column.

14.20.267. Logit

Category

General-Purpose Functions

Description

Inverse Logistic Function

Syntax

Logit(value)

Parameters

value - formula

Notes

The Logit function transforms probabilities from the range [0, 1] back to the real number line (-∞, ∞). In trading, it can be used to revert normalized probability-based data to meaningful market values, aiding in the interpretation of logistic regression models.

Logit(value) is calculated as *Log(value / (1 - value)).*

Example

```
LogitVal: Logit(0.75)
```

Inverse logistic: maps probability 0.75 to log(0.75/0.25) = log(3).

14.20.268. 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.

14.20.269. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
Low52: Lowest(L, 252)
```

Lowest and LLV are interchangeable names.

14.20.270. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
MA200: MA(C, 200)
```

MA and Avg are interchangeable names.

14.20.271. MACD

Category

Indicator Functions

Description

Moving Average Convergence Divergence

Syntax

MACD(len1, len2)

Parameters

len1 - faster moving average period (typically 12)

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 **one-pass calculation** when used in the **Data Section** with non-variable lengths.

Examples

```
MyMACD: MACD(12, 26)
EntrySetup: Cross(MyMACD, 0)
```

Enter when MACD crosses above the zero line.

14.20.272. MACDH

Category

Indicator Functions

Description

MACD Histogram

Syntax

MACDH(len1, len2, len3)

Parameters

len1 - faster moving average period (typically 12)

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 **one-pass calculation** when used in the **Data Section** with non-variable lengths.

Example

```
Hist: MACDH(12, 26, 9)
```

Histogram > 0 when MACD is above its signal line.

14.20.273. MACDS

Category

Indicator Functions

Description

MACD Signal Line

Syntax

MACDS(len1, len2, len3)

Parameters

len1 - faster moving average period (typically 12)

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 **one-pass calculation** when used in the **Data Section** with non-variable lengths.

Examples

```
MyMACD: MACD(12, 26)
Signal: MACDS(12, 26, 9)
EntrySetup: Cross(MyMACD, Signal)
```

Enter when MACD crosses above its signal line.

14.20.274. 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. For example if your broker charges 1.5% more than the fed funds rate for margin loans then specify *MarginIntPct: 1.5*.

If *MarginIntPct* is provided when there is no *RiskFreeRateSym* setting then *MarginIntPct* is simply the annualized margin interest rate, e.g. *MarginIntPct: 6.5*.

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 interest rate received for positive excess cash.

14.20.275. 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 reported values of **S.Invested** and **S.Exposure** (open-position mark-to-market is added to invested/exposure only when MarkToMarket is True), as well as the internal calculations of **S.DDDlr**, **S.DDPct**, **S.MaxDDDlr**, **S.MaxDDPct**, **S.NetDlr** and **S.NetPct** (and their **Min**/**Max** variants, which equal the base stats when MarkToMarket is False). It also affects the default value of **Allocation** if no formula is provided, since that default is *Combined(S.Equity)*.

14.20.276. 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*")*

Example

```
IsETF: Match(?Name, "*ETF*")
```

Wildcard pattern matching: ? matches one character, * matches zero or more.

14.20.277. 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.

Example

```
TrueRange: Max(H - L, Abs(H - C[1]), Abs(L - C[1]))
```

True range: the greatest of range, high-gap, or low-gap.

14.20.278. 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).

Example

```
SecondBest: MaxN(2, O, H, L, C)
```

The second-highest of today's OHLC values.

14.20.279. MaxCatExp

Category

Strategy Elements

Description

Percent exposure limit for positions in the same category for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MaxCatExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its position size plus those of other setups and open positions in that category would not exceed the percent of **S.Alloc** expressed as *MaxCatExp*.

Use the **Category** formula to define the category of any security.

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 category exposure" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxCatExp* is not specified then there is no limit placed on same-category exposure level, though there might still be same-category constraints if the strategy specifies **MaxCatInv** or **MaxSameCat**.

Note that opening gaps beyond a specified order price may cause actual same-category exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended exposure to the same category".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.280. MaxCatInv

Category

Strategy Elements

Description

Dollar investment limit for positions in the same category for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MaxCatInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its position size plus those of other setups and open positions in that category would not exceed the dollar value expressed as *MaxCatInv*.

Use the **Category** formula to define the category of any security.

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 category invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxCatInv* is not specified then there is no limit placed on same-category investment, though there may still be same-category constraints if the strategy specifies **MaxCatExp** or **MaxSameCat**.

Note that opening gaps beyond a specified order price may cause actual same-category investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended investment in the same category".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.281. 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 specialized formula that is only needed when modeling strategies that allow more setups than entries. It should NOT be used as a general-purpose strategy constraint.

Use **MaxNewPos** if you want to cap the daily new position count at order time.

See also **EntryScore**.

14.20.282. 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.

A setup is only selected if its position size plus those of other setups selected so far plus current **S.Invested** of open positions divided by current **S.Alloc** would not exceed the percentage expressed as *MaxExposure*.

Note that the meaning of the cap depends on the strategy's **S.Alloc**: for a default strategy this is the combined account value, while for a strategy using **StartPercent** or a custom **Allocation** it is that strategy's own allocation. The same *MaxExposure: 100* therefore represents different dollar amounts in the two cases.

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 total percent exposure, though there might still be investment limits if the strategy specifies **MaxInvested** and/or **MaxPositions**.

To cap exposure at the strategy's current allocation use *MaxExposure: 100*.

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**

14.20.283. MaxInvested

Category

Strategy Elements

Description

Dollar 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.

A setup is only selected if its position size plus those of other setups selected so far plus current **S.Invested** of open positions would not exceed the dollar value expressed as *MaxInvested*.

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 total 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*, or more simply *MaxExposure: 100*.

Note that opening gaps beyond a specified order price may cause actual investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended investment".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.284. MaxLongExp

Category

Strategy Elements

Description

Open long position exposure limit for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MaxLongExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A long setup is only selected if its position size plus those of other long setups selected so far plus current **S.LongInv** of open positions divided by current **S.Alloc** would not exceed the percentage expressed as *MaxLongExp*.

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 long exposure" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxLongExp* is not specified then there is no limit placed on long percent exposure, though there might still be a long investment limit if the strategy specifies **MaxLongInv**.

Note that opening gaps beyond a specified order price may cause actual long exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended long exposure".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.285. MaxLongInv

Category

Strategy Elements

Description

Dollar investment limit in long positions for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MaxLongInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A long setup is only selected if its position size plus those of other long setups selected so far plus current **S.LongInv** of open positions would not exceed the dollar value expressed as *MaxLongInv*.

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 long invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxLongInv* is not specified then there is no limit placed on long dollar investment level, though there might still be a long investment limit if the strategy specifies **MaxLongExp**.

Note that opening gaps beyond a specified order price may cause actual long investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended long investment".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.286. MaxNetExp

Category

Strategy Elements

Description

Maximum net long-short position exposure for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MaxNetExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its signed position size plus those of other setups selected so far plus current net **S.LongInv** - **S.ShortInv** of open positions divided by current **S.Alloc** would not exceed the percentage expressed as *MaxNetExp*. (In practice only a long setup could trigger this constraint.)

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 net exposure" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxNetExp* is not specified then there is no upper limit placed on net percent exposure, though there might still be a net investment upper limit if the strategy specifies **MaxNetInv**.

Note that opening gaps beyond a specified order price may cause actual net exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended net exposure".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.287. MaxNetInv

Category

Strategy Elements

Description

Maximum net long-short dollar investment for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MaxNetInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its signed position size plus those of other setups selected so far plus current net **S.LongInv** - **S.ShortInv** of open positions would not exceed the dollar value expressed as *MaxNetInv*. (In practice only a long setup could trigger this constraint.)

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 net invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxNetInv* is not specified then there is no upper limit placed on net dollar investment level, though there might still be a net upper limit if the strategy specifies **MaxNetExp**.

Note that opening gaps beyond a specified order price may cause actual net investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended net investment".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.288. 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**

14.20.289. 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**

14.20.290. 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**

14.20.291. MaxPerTurn

Category

Strategy Elements

Description

Defines how many setups per turn a strategy can select 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).

Selecting a setup means adding that setup to the list of tomorrow's entry orders.

Selection turns take place within a single entry phase - only strategies that share a phase compete

turn by turn. Use **EntryPhase** when one strategy's setups should be finalized entirely before another's are considered.

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.292. 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**

14.20.293. 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.

Use the **Category** formula to define the category of any security.

Setups skipped for this reason display "max same category" 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**

14.20.294. MaxSameSym

Category

Strategy Elements

Description

Count limit for positions in the same symbol for a strategy or group of strategies

Input

Any formula specifying a count

Notes

MaxSameSym 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 in that symbol would not exceed the value returned by this formula.

Setups skipped for this reason display "max same symbol" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

The default *MaxSameSym* for each individual **Strategy** is 1, i.e., by default no strategy can hold more than one position in the same symbol.

For **StatsGroup** or **Combined**, the default is "unlimited", though there might still be same-symbol limits if the strategy specifies **MaxSymExp** and/or **MaxSymInv**.

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.295. 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.*

The only common use case for this constraint is to make tests run faster by trimming the setup list before processing the setups.

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.296. MaxShortExp

Category

Strategy Elements

Description

Open short position exposure limit for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MaxShortExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A short setup is only selected if its position size plus those of other short setups selected so far plus current **S.ShortInv** of open positions divided by current **S.Alloc** would not exceed the percentage expressed as *MaxShortExp*.

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 short exposure" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxShortExp* is not specified then there is no limit placed on short percent exposure, though there might still be a short investment limit if the strategy specifies **MaxShortInv**.

Note that opening gaps beyond a specified order price may cause actual short exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended short exposure".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.297. MaxShortInv

Category

Strategy Elements

Description

Dollar investment limit in short positions for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MaxShortInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A short setup is only selected if its position size plus those of other short setups selected so far plus current **S.ShortInv** of open positions would not exceed the dollar value expressed as *MaxShortInv*.

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 short invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxShortInv* is not specified then there is no limit placed on short dollar investment level, though there might still be a short investment limit if the strategy specifies **MaxShortExp**.

Note that opening gaps beyond a specified order price may cause actual short investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended short investment".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.298. MaxSymExp

Category

Strategy Elements

Description

Percent exposure limit for positions in the same symbol for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MaxSymExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its position size plus those of other setups and open positions in that symbol would not exceed the percent of **S.Alloc** expressed as *MaxSymExp*.

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 symbol exposure" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxSymExp* is not specified then there is no limit placed on same-symbol exposure level, though there might still be same-symbol constraints if the strategy specifies **MaxSymInv** or **MaxSameSym**.

Note that opening gaps beyond a specified order price may cause actual same-symbol exposure to exceed the specified cap in some situations -- think of this constraint as "maximum intended exposure to the same symbol".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.299. MaxSymInv

Category

Strategy Elements

Description

Dollar investment limit for positions in the same symbol for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MaxSymInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its position size plus those of other setups and open positions in that symbol would not exceed the dollar value expressed as *MaxSymInv*.

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 symbol invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MaxSymInv* is not specified then there is no limit placed on same-symbol investment, though there may still be same-symbol constraints if the strategy specifies **MaxSymExp** or **MaxSameSym**.

Note that opening gaps beyond a specified order price may cause actual same-symbol investment to exceed the specified cap in some situations -- think of this constraint as "maximum intended investment in the same symbol".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.300. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Example

```
MinusDI: MDI(14)
PlusDI: PDI(14)
```

Typically used together with PDI to determine directional trend.

14.20.301. 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 (or the average of the two middle values when count is even).

Example

```
MedVol: Median(V, 50)
```

Median is less sensitive to outliers than MA.

14.20.302. 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 1

length - the number of characters to include

Example

```
MidPart: Mid(?Symbol, 1, 2)
```

Extract 2 characters starting at position 1 (the first two characters).

14.20.303. 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.

Example

```
Support: Min(L, L[1], L[2])
```

Lowest of the last 3 bars' lows.

14.20.304. 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).

Example

```
SecondLow: MinN(2, O, H, L, C)
```

The second-lowest of today's OHLC values.

14.20.305. MinFreeCash

Category

Strategy Elements

Description

Minimum free cash to keep on hand in the trading account

Input

Any formula specifying a dollar amount

Notes

MinFreeCash is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if current **S.FreeCash** minus cash required to enter this position and other setups selected so far would not go below *MinFreeCash*.

For a futures, FX, or crypto position the cash required to enter is the number of contracts to enter times the **S.Collateral** (collateral) required per contract.

For a long stock position the cash required to enter is the number of shares to buy times the expected entry price.

For a short stock position cash is received upon entry so a short setup cannot trigger this constraint and will make it less likely for a subsequent long setup to do so.

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 "min free cash" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MinFreeCash* is not specified and **AccountType** is *Margin* (the default) then there is no lower limit placed on free cash in the account. If *AccountType* is *CashSameDay* or *CashNextDay* then the default *MinFreeCash* is 0 (cannot go negative, i.e., cannot use margin).

To allow a limited amount of margin rather than unlimited, specify a negative value. For example, *MinFreeCash: -0.5 * S.Alloc* permits borrowing up to 50% of the current allocation.

Note that opening gaps beyond a specified order price may cause actual free cash to drop below the specified minimum in some situations -- think of this constraint as "minimum intended free cash". Allow a bit of extra room in your **MaxExposure** level if this is an issue.

See Also: **Backtest Engine Details** and **Capacity Constraints**

Dividends accrued to open positions are not counted as free cash until the position exits, so by default they cannot satisfy this constraint on their ex-date. The **DividendsToCash** setting changes that.

14.20.306. MinNetExp

Category

Strategy Elements

Description

Minimum net long-short position exposure for a strategy or group of strategies

Input

Any formula specifying a percentage

Notes

MinNetExp is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its signed position size plus those of other setups selected so far plus current net **S.LongExp** - **S.ShortExp** of open positions divided by current **S.Alloc** would not be less than the percentage expressed as *MinNetExp*. (In practice only a short setup could trigger this constraint.)

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 "min net exposure" in the *Reason* column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MinNetExp* is not specified then there is no lower limit placed on net percent exposure, though there might still be a net investment lower limit if the strategy specifies **MinNetInv**.

Note that opening gaps beyond a specified order price may cause actual net exposure to fall below the specified minimum in some situations -- think of this constraint as "minimum intended net exposure".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.307. MinNetInv

Category

Strategy Elements

Description

Minimum net long-short dollar investment for a strategy or group of strategies

Input

Any formula specifying a dollar amount

Notes

MinNetInv is one of the **Capacity Constraints** formulas used in the setup selection process.

A setup is only selected if its signed position size plus those of other setups selected so far plus current net **S.LongInv** - **S.ShortInv** of open positions would not be less than the dollar value expressed as *MinNetInv*. (In practice only a short setup could trigger this constraint.)

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 "min net invested" in the Reason column of the **Trade List**, provided that the **KeepTrades** setting included *Skipped*.

If *MinNetInv* is not specified then there is no lower limit placed on net dollar investment level, though there might still be a net lower limit if the strategy specifies **MinNetExp**.

Note that opening gaps beyond a specified order price may cause actual net investment to fall short of the specified minimum in some situations -- think of this constraint as "minimum intended net investment".

See Also: **Backtest Engine Details** and **Capacity Constraints**

14.20.308. MissingBars

Category

General-Purpose Functions

Description

Number of bars missing before the current bar in this symbol

Syntax

MissingBars(symbol {$CalSym})

Parameters

symbol - the calendar symbol to use for comparison (optional)

Notes

A "missing bar" is a bar that is absent in this stock while being present in most other stocks from the same exchange.

This function returns a count of missing bars preceding the current bar. (By definition the current bar must be present.)

For most symbols on most dates, *MissingBars()* will return 0.

A missing bar is detected when the comparison symbol has a bar for that date but the current symbol does not.

The default comparison symbol is the **CalendarSym** for the current strategy, or the global date list (every date with a bar for any symbol) if no CalendarSym is set.

The optional argument can be used to specify a different comparison symbol if desired.

See also the **AllowMissingBar** strategy setting, which determines whether trades can occur on a bar for which *MissingBars()* returns > 0.

14.20.309. 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.

14.20.310. MostRecent

Category

General-Purpose Functions

Description

Evaluate expression as of the most recent available bar when one is missing on the current date

Syntax

MostRecent(expr)

Parameters

expr - any expression (typically with Extern)

Notes

To evaluate an expression for a symbol on a given date, that symbol must have a bar on that date. The evaluation date is the date of the current symbol's current bar at the moment of evaluation. When the symbol has no bar on that date, the lookup cannot be anchored and the result is NaN. This arises in two common situations:

- Cross-symbol references on a holiday. Suppose the current symbol is AUDUSD and the date is April 3, 2026 (Good Friday). US and AU equity markets are closed that day, so *Extern($SPY, Close)* returns NaN even though AUDUSD itself has a bar.
- Higher-timeframe scripts reading a lower timeframe. RealTest labels each weekly bar with the Friday of that week whenever any symbol in the data file traded that Friday - the "global calendar." A symbol that was closed that Friday has no daily bar on the weekly bar's label date, so *Extern(~Daily, MA(C, 10))* returns NaN for that symbol even though its weekly bar exists.

MostRecent(expr) resolves both cases. When the bar for the current context is missing, it evaluates *expr* as of the most recent earlier bar instead of returning NaN. The expression is evaluated normally as of that bar - MostRecent does not fabricate or repeat data, so *MA(C, 10)*, for instance, is the average of the ten real bars ending on that earlier date. (This differs from *Padding: AllMarketDays*, which inserts a synthetic bar that repeats the prior close and can distort range-based indicators such as ATR.)

Using the examples above, *MostRecent(Extern($SPY, Close))* returns SPY's Thursday close, and *MostRecent(Extern(~Daily, MA(C, 10)))* returns the 10-day average close as of Thursday.

MostRecent must enclose the Extern, not the reverse. *Extern($SPY, MostRecent(Close))* still returns NaN on a holiday, because Extern performs its bar lookup - and fails - before its argument is evaluated, so the inner MostRecent never takes effect. As a rule, place MostRecent around whatever is returning NaN.

The same rule applies to barsize-specific Data arrays. Reading a lower-timeframe Data item from a higher-timeframe context performs the same cross-barsize lookup as Extern(~Barsize, …) - an implied Extern at the point of reference. Apply MostRecent where you reference the item, not inside its definition (where there is no Extern to enclose and it has no effect). Alternatively, define the item with an explicit Extern(~Barsize, …) and wrap that with MostRecent, so the fallback value is computed once and every reference uses it.

Examples

Cross-symbol reference that survives a holiday:

```
Data:
      spyClose: MostRecent(Extern($SPY, Close))
```

Daily indicator in a weekly-barsize script - preferred form, fallback stored once:

```
Settings:
      BarSize: Weekly
Data:
```

```
    dailyMA10: MostRecent(Extern(~Daily, MA(C, 10)))
```

Same result using a barsize-specific Data item - wrap the reference, not the definition:

```
Settings:
    BarSize: Weekly
Data:
    BarSize: Daily
    ma10: MA(C, 10)              // calculated on daily bars
Scan:
    x: MostRecent(ma10)           // apply MostRecent where the weekly context
reads it
```

The following does not work - MostRecent inside a barsize-specific definition has no Extern to enclose:

```
Data:
    BarSize: Daily
    ma10: MostRecent(MA(C, 10))  // returns NaN on the missing date
```

14.20.311. Namespace

Category

Script Sections

Description

Specifies a prefix for user-defined items, to prevent duplication when including multiple scripts

Syntax

Namespace: xxx

Notes

All names defined after a *Namespace* statement are internally converted to have the specified prefix.

For example, *Data: Universe: InSPX* would internally become *Data: xxx.Universe: InSPX*.

Subsequent statements, while the same *Namespace* is still in effect, can then refer to *Universe* as EITHER *Universe* or *xxx.Universe*.

However, if the above *Data* statement was in an **Included** script, references in the *including* script would be required to use *xxx.Universe*.

An item can be referred to by its bare name only while its *Namespace* is in effect. From a different *Namespace*, after *Namespace: none*, or from an including script, it must be referred to by its full *xxx.name*. An item defined with no *Namespace* has no prefix, so it cannot be referred to from within a *Namespace*.

To share one value across several namespaces, define it under its own *Namespace* and refer to it by its full name. For example, define *Param1* under *Namespace: Globals*, then refer to it elsewhere as *Globals.Param1*.

The most common use case for *Namespace* is e.g.:

Namespace: ns1

Include: myscript1.rts

Namespace: ns2

Include: myscript2.rts

// etc.

This would ensure that all names defined in each included script remain unique.

Use *Namespace: none* to remove the current *Namespace* and revert to not auto-prefixing the item names.

A *Namespace* prefix, when specified, is applied to all **User-Defined Item** names.

Namespace is also applied to these special-use names:

- **Import IncludeList** user-defined names, e.g. *IncludeList: SPY,TLT,GLD {"my_etfs"}* would internally be named *xxx.my_etfs*
- **InList** references to the above, e.g., *InList("my_etfs"),* would internally become *InList("xxx.my_etfs")*
- Names derived by the **Item** function, e.g., *Item("factor{#}", factor_number)* would return e.g. *xxx.factor3*

Another use case for *Namespace* is to prevent **Item Name Overrides**. By default, in most script sections, RealTest will replace the formula associated with a name with the new one found under the same name. Use *Namespace* to ensure that both formulas are preserved and given unique names.

Finally, in many cases, it may be preferable to simply use unique item names across all of your scripts, rather than relying on *Namespace* to differentiate them.

See also **Repeat Blocks** and **Combining Scripts**.

14.20.312. 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.

Example

```
EntrySkip: NextOpen < C * 0.95
```

Skip entry if the open gaps down more than 5%. NextOpen is known at the time of entry.

14.20.313. NoNan

Category

General-Purpose Functions

Description

Evaluate an expression with no possibility of a 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).

Example

```
SafeRatio: NoNan(C / V, 0)
```

Returns 0 instead of NaN when volume is zero.

14.20.314. 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.

14.20.315. 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.

14.20.316. NthOfMonth

Category

General-Purpose Functions

Description

Whether current bar date is nth specific weekday of current month

Syntax

NthOfMonth(weekday, nth, market {0})

Parameters

weekday - 1 = Monday, 2 = Tuesday, ..., 7 = Sunday

nth - 1 = first from start, -1 = first from end, 2 = second from start, -2 = second from end, ...

market - if 0 (default), occurrences of the weekday are counted using all calendar dates; if 1, only market (trading) days are counted, so an occurrence that falls on a market holiday is skipped

Return Value

1 if the current bar's day of week is the specified weekday and it is the nth instance of that weekday in the current month, else 0.

(Again, where nth means from the start if positive, or from the end if negative.)

Notes

This function applies to the date of **the current bar** being evaluated.

The current bar must be the specified weekday for the result to be true.

The other instances of that weekday that determine whether the current bar is the "nth" do NOT have to be market days.

For example Monday September 9 is the second Monday in US markets even though the 2nd was a holiday (labor day).

Examples

```
IsThirdFriday: NthOfMonth(5, 3)
IsFinalTuesday: NthOfMonth(2, -1)
```

To find a past or future NthOfMonth date, use e.g.

```
PrevFirstMondayDate: WhenTrue(NthOfMonth(1, 1), BarDate)
NextThirdFridayDate: WhenTrue(NthOfMonth(5, 3), BarDate, 0, -1)
```

Note that future NthOfMonth values cannot be determined beyond the last date of the current data file, i.e., they are only usable in backtests, not in order generation.

14.20.317. NumBars

Category

Settings

Description

Number of data bars to include in a scan or test

Notes

Can be used alone or in conjunction with **StartDate** or **EndDate**. The date window is resolved from whichever end is anchored, and *EndDate* takes precedence:

With any *EndDate* - a literal date, or *Latest* - the window is *NumBars* bars counting back from that date, and a *StartDate* is ignored however it was set. With a *StartDate* and no *EndDate*, it is *NumBars* bars counting forward from that date; *StartDate: Earliest* gives the first *NumBars* bars of the data file. With *NumBars* alone - no date settings at all - it is the most recent *NumBars* bars.

Since a **ScanSettings** section inherits the **Settings** section, a script whose test uses a dated *StartDate* and *EndDate* can still scan the most recent bars by putting *EndDate: Latest* and *NumBars* in *ScanSettings* - the usual daily-scan idiom.

When **BarSize** is Daily, *NumBars* is a count of market days. Otherwise it is a count of bars at that *BarSize* (weeks, months, quarters, or years).

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
```

14.20.318. 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 (*Sum((C > C[1]) * V, len) - Sum((C < C[1]) * V, len)) / Sum(V, len) * 100*.

Example

```
MyOBV: OBV(20)
```

14.20.319. 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.

14.20.320. 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 a limit or stop entry order 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. A stop or target exit that gaps through at the open is still treated as a stop or target exit, so **Slippage** (not *OpenSlip*) is applied to it.

14.20.321. OptAnchor

Category

Settings or **Optimize Settings**

Input

None, *Start*, or *End* (default is *None*)

Description

When date intervals are used (see **OptTimeUnit**), each interval is by default a sliding window of the same length. OptAnchor optionally locks one end of the range: anchoring the *Start* date makes each interval longer than the preceding one, and anchoring the *End* date does the opposite.

14.20.322. OptByStrategy

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Runs the optimization separately for each strategy, as if every other strategy were commented out -- producing one optimization run (and one set of results) per strategy. Only affects Optimize mode. Can be combined with **OptBySymbol** to produce strategy × symbol optimizations.

When parameters are being optimized (in Combinatorial, Random or Genetic mode), each parameter combination is generated once and run for every strategy and symbol before the next combination is generated. In Random and Genetic modes every strategy and symbol therefore evaluates the same set of combinations rather than independently drawn ones.

14.20.323. OptBySymbol

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Splits the job into one test (or one whole optimization) per symbol, as if every other symbol were absent from the data. Can be combined with **OptByStrategy**.

When parameters are being optimized (in Combinatorial, Random or Genetic mode), each parameter combination is generated once and run for every strategy and symbol before the next combination is generated. In Random and Genetic modes every strategy and symbol therefore evaluates the same set of combinations rather than independently drawn ones.

14.20.324. OptClearResults

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *True*)

Description

When *True* (the default), the active results window is emptied before the optimization starts. When *False*, the new test rows are appended to the existing list.

14.20.325. OptContinueResults

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Resumes a previously interrupted optimization where it left off. RealTest seeds the run from the parameter combinations already present in the **ResultsFile** RTR and skips those combinations during enumeration.

Notes

Requires Combinatorial mode, a ResultsFile setting, no OptKeepBest pruning, no Date Intervals, no For Each Strategy / Symbol, and no Walk-Forward. An optimization with a ResultsFile setting saves its RTR frequently during the run, so a crash, power loss, or manual stop loses at most a few minutes of completed tests.

14.20.326. OptimizeMode

Category

Settings or **Optimize Settings**

Input

Combinatorial, *Sequential*, *Genetic*, *Random*, or *Defaults* (default is *Combinatorial*)

Description

Chooses how the parameter space is searched. The **Multiple Tests and Optimization** topic describes each mode in detail; in brief:

- *Combinatorial* tests every combination of the checked parameter values (exhaustive nested loop).
- *Sequential* sweeps one parameter at a time, locking in each parameter's best-scoring value before moving to the next.
- *Genetic* randomly mutates a subset of parameters each step (see **OptMutatePct**), keeping changes that improve the score.
- *Random* rerolls every parameter before each test and ignores the score - useful for studying the distribution of outcomes.
- *Defaults* (shown as "Defaults Only" in the dialog) runs once using each parameter's default value - equivalent to unchecking every parameter.

Notes

Genetic and Random modes require **OptIterations** to set how many tests to run. Sequential and Genetic searches are driven by **OptScoreCol**.

14.20.327. OptimizeSettings

Category

Script Sections

Description

Defines the settings to use only when the script **run mode** is **Optimize**.

Notes

The general-purpose **Settings** section is always applied first, then modified by any items specified in *OptimizeSettings* when applicable.

See **Settings Sections** and **Optimize Settings** for details.

See also the **Multiple Tests and Optimization** topic.

This section doesn't have to be written by hand: the **Save to Script** button in the **Optimizer dialog** creates or updates it from the dialog's current settings.

14.20.328. OptIterations

Category

Settings or **Optimize Settings**

Input

a number (default is *1*)

Description

The number of times to run the configured test. Use it to repeat a test that uses the **Random** function (to obtain a range of potential outcomes), or to set how many tests **Genetic or Random mode** should run.

14.20.329. OptKeepBest

Category

Settings or **Optimize Settings**

Input

a number (default is *0*, meaning keep all)

Description

Keeps only the N highest-scoring tests during the run, discarding the rest as the optimization progresses. This makes it practical to explore a very large parameter space while retaining only the best results.

Notes

Requires **OptSortResults**. Use **OptKeepByDate** or **OptKeepBySymbol** to apply the limit per date interval or per symbol.

14.20.330. OptKeepByDate

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Applies **OptKeepBest** separately to each date interval rather than across the whole run. Requires **OptSortResults** and a non-None **OptTimeUnit**.

14.20.331. OptKeepBySymbol

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Applies **OptKeepBest** separately to each **OptBySymbol** test. Requires **OptSortResults** and OptBySymbol.

14.20.332. OptKeepNoTrades

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

By default, a parameter combination that produces zero trades is discarded from the results. Set

this to *True* to keep those rows.

14.20.333. OptMultiSaveAs

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

When *True*, performs **SaveStatsAs** and/or **SaveTradesAs** once per test, creating a separate file for each. Those settings are otherwise ignored in a multi-test run.

14.20.334. OptMutatePct

Category

Settings or **Optimize Settings**

Input

a number (default is *25*)

Description

For **Genetic mode**, the mutation rate - the probability (in percent) that each parameter is reset to a new random value from its list before each test. Larger values explore more aggressively; smaller values refine around the current best.

14.20.335. OptNoDialog

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

When *True*, RealTest runs the optimization without first displaying the Optimize dialog - useful for unattended or **batch-mode** runs where every setting is specified in the script. This is the one optimize setting that has no equivalent control in the dialog.

14.20.336. OptNoStats

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Drops the per-period statistics array from each saved test, keeping only the precomputed Results

column values. This dramatically reduces RTR size and memory use, making optimizations of millions of tests feasible.

Notes

Rows saved this way can no longer drive the Graph, Plot, Report, Save Stats, Save Trades, or Save Positions commands - only their Results column values remain available.

14.20.337. OptOverfit

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Runs an **Overfitting Analysis** when the optimization completes: the CSCV/PBO Overfitting Score (how often the in-sample best parameter combination held up out-of-sample) and the Walk-Forward Correlation (how well the whole parameter surface carried from in-sample to out-of-sample). The block count is set with **OptOverfitBlocks**.

Cannot be combined with Walk-Forward, Date Intervals, For Each Strategy / Symbol, Shard Processes, or Continue From File. In command line batch mode the analysis also writes *overfit.csv* to the OutputPath folder.

14.20.338. OptOverfitBlocks

Category

Settings or **Optimize Settings**

Input

an even number from *4* to *20* (default is *16*)

Description

The number of equal time blocks (S) into which the test period is cut for the **Overfitting Analysis** enabled by **OptOverfit**. The number of in-sample/out-of-sample splits is the number of ways to choose S/2 blocks from S: 70 for 8 blocks, 924 for 12, 12,870 for 16, 184,756 for 20.

Keep the default 16 for ten or more years of daily data; drop to 12 or 8 for shorter histories so that each block still contains enough days (a warning appears below 20 days per block). When comparing strategies, compare at the same block count.

14.20.339. OptScoreCol

Category

Settings or **Optimize Settings**

Input

a Results column name

Description

Names the **Results** column to use as the optimization's fitness function. Because Results columns are formula-based, you can score by any expression - net profit, Sharpe, MAR, or any custom formula you define.

Notes

The score column ranks rows when **OptSortResults** is on, drives the Sequential and Genetic searches, and selects the best parameter sets when building a walk-forward test.

14.20.340. OptShards

Category

Settings or **Optimize Settings**

Input

a number, 2 to 64 (default is *0*; 0 or 1 = no sharding)

Description

For very large **Combinatorial** optimizations, OptShards divides the parameter space across multiple parallel RealTest processes (2 to 64). Each shard runs an independent slice of the largest active parameter's value list and writes its own RTR file; when all shards finish, RealTest merges them into a single combined RTR.

Notes

Sharding requires Combinatorial, Genetic, or Random mode (not Sequential), no Date Intervals / For Each Strategy / For Each Symbol / Walk-Forward, and more total tests than shards. If there is nothing meaningful to slice across, the optimization runs inline in a single process, so OptShards is safe to leave set in scripts that may or may not need it.

In Combinatorial mode, sharding can be combined with **OptContinueResults** to resume a stopped sharded run, provided the same shard count is used.

14.20.341. OptShowResults

Category

Settings or **Optimize Settings**

Input

Live, *AtEnd*, or *None* (default is *Live*)

Description

Live updates the results window as each test finishes (the historical behavior). *AtEnd* runs the optimization headless and opens the RTR file when the run completes. *None* runs headless and never opens the file - useful for batch pipelines where the RTR is the final artifact.

Notes

AtEnd and *None* require a **ResultsFile** setting and skip the per-test list update, which materially speeds up very large optimizations.

14.20.342. OptSortResults

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Works together with **OptKeepBest**: after each test completes, the result rows are sorted by **OptScoreCol** and all but the top-scoring rows are pruned, so only the best tests are retained as the run progresses. On its own, without a positive OptKeepBest value, this setting has no effect.

14.20.343. OptSummaryLog

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

When *True*, a summary log of the optimization is produced when the run finishes. If a ResultsFile is set, the summary is written to a matching ".log" file beside it (the ResultsFile path with its .rtr extension replaced by .log) - the deliverable in batch mode; interactively, that log is also opened in a log window.

14.20.344. OptTestInterval

Category

Settings or **Optimize Settings**

Input

a number (default is *1*)

Description

The spacing between successive date intervals, expressed in **OptTimeUnit** units. Used with OptTimeUnit and **OptTestLength** to define multi-interval optimizations and walk-forward tests.

14.20.345. OptTestLength

Category

Settings or **Optimize Settings**

Input

a number (default is *1*)

Description

The length of each date interval, expressed in **OptTimeUnit** units. With the default interval and length of 1, each step advances by one time unit and covers one time unit.

14.20.346. OptTimeUnit

Category

Settings or **Optimize Settings**

Input

None, *Days*, *Weeks*, *Months*, or *Years* (default is *None*)

Description

The time unit for multi-interval optimizations or walk-forward tests. With *None* no date intervals are used. Setting it to *Years* is a convenient way to produce one result row per calendar year of a date range.

Notes

Used together with **OptTestLength**, **OptTestInterval**, **OptAnchor**, and **OptWalkForward**.

14.20.347. OptWalkForward

Category

Settings or **Optimize Settings**

Input

True or *False* (default is *False*)

Description

Generates a **walk-forward** test from the configured date intervals and parameter selection.

Notes

Requires a non-None **OptTimeUnit** and **OptKeepByDate**, and is not combined with **OptByStrategy** or **OptBySymbol**.

14.20.348. OrderClerkFolder

Category

Settings or **Order Settings**

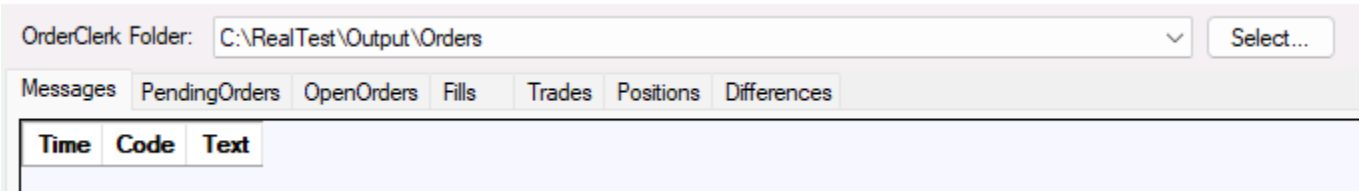
Description

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*, *OrderClerkSync*, or *OrderClerkStats*.

This will be the same folder that is selected at the top of the OrderClerk main window:



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.

14.20.349. OrderNote / OrderExtra

Category

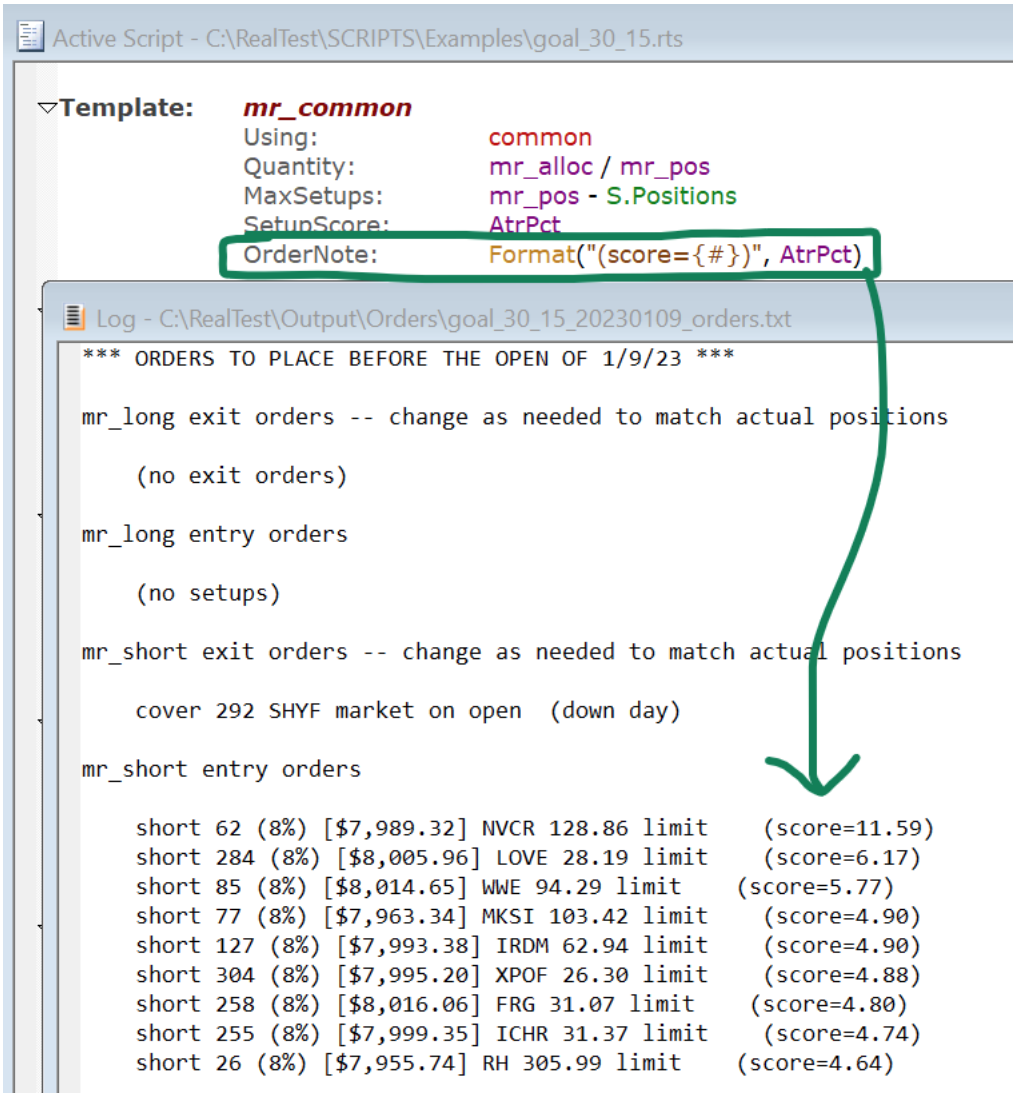
Strategy Elements

Description

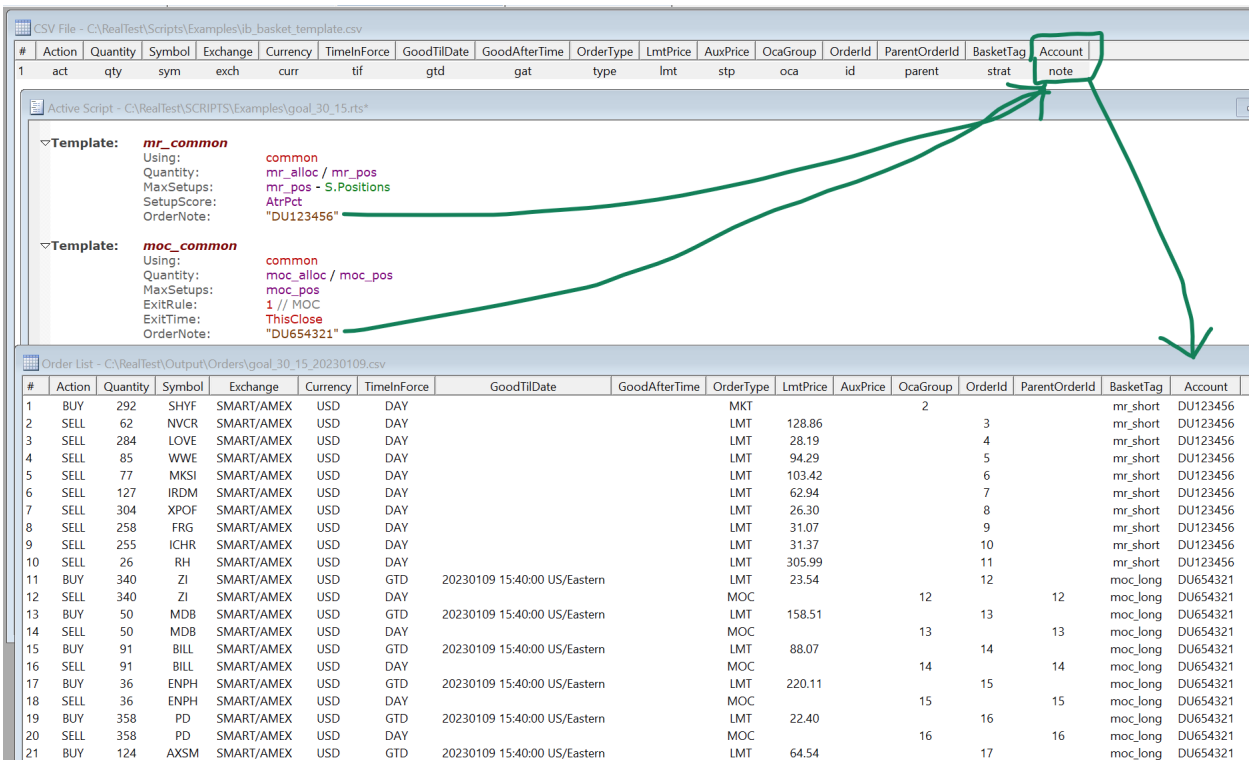
String to add to the text of each order and as the "note" value in a **CSV Order Basket**

Notes

Here is an example of how this works in a text order list:



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:



In addition to *OrderNote*, up to nine other strategy-specific values can be added to an orders template.

The strategy formula names for these are *OrderExtra1*, *OrderExtra2, ...OrderExtra9*.

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, ... extra9*.

14.20.350. 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.

14.20.351. OrderRank

Category

Current Position Information

Description

Returns the order rank number for this position's entry across all strategies.

Notes

The order ranking mechanism can be observed in detail by running a test with both *KeepTrades: Skipped* and *TestOutput: Log* enabled.

14.20.352. OrdersClassDelim

Category

Settings or **Order Settings**

Description

Specifies how a symbol class suffix (e.g. the .B in BRK.B) is written in generated order symbols

Choices

Space - (default) replace the dot with a space ("BRK B"), the Interactive Brokers convention

Dot - leave the symbol unchanged ("BRK.B")

Dash - replace the dot with a dash ("BRK-B")

Slash - replace the dot with a slash ("BRK/B"), the Schwab/Fidelity convention

None - remove the dot ("BRKB")

Notes

This setting applies only when the symbol's **ExchangeMap** row was matched by exchange name. When the row was matched by a dot wildcard pattern such as *.AU, the suffix is a country identifier rather than a symbol class, and it is removed from the order symbol as always (e.g. BHP.AU becomes BHP) regardless of this setting.

Since RealTest symbols cannot contain spaces, the default class suffix conversion is also the way to produce order symbols with embedded spaces. For example, with a Helsinki stock whose broker symbol is "NDA FI", an **IncludeList** alias of NDA.FI causes the order symbol "NDA FI" to be emitted when the symbol's exchange row matches by name.

Example

```
OrdersClassDelim: Slash
```

Writes class-share symbols in the Schwab/Fidelity convention, e.g. BRK/B.

14.20.353. OrdersComment

Category

Settings or **Order Settings**

Description

An optional string to add to the top of the order list when orders are generated.

Notes

The *OrdersComment* setting is interpreted as a formula.

Use the **Format** function to build a string that displays whatever information from your script variables would be useful to see in an order list.

Note that formula output strings are limited to 80 characters in length.

14.20.354. OrderSettings

Category

Script Sections

Description

Defines the settings to use only when the script **run mode** is **Orders**.

Notes

The general-purpose **Settings** section is always applied first, then modified by any items specified in *OrderSettings* when applicable.

See **Settings Sections** for details.

14.20.355. OrdersFile

Category

Settings or **Strategy Elements** or **Order Settings**

Description

Path and name of the order list file to create when generating orders.

Notes

If *OrdersFile* is not specified then the default path is c:\RealTest\Output\Orders (assuming RealTest is installed in c:\RealTest) and the default file is *script_date*_orders.txt (where *script* is the current script name and *date* is the order date).

When **OrdersMode** is anything other than the default *Text* mode, two orders files are created: *script_date*_orders.txt and *script_date*_orders.csv in OrderClerk modes or *script_date*.csv in Rebalance or Template mode.

When *OrdersFile* is specified, the file name should end in .txt for the default *OrdersMode* or in .csv in any other mode.

A settings-level *OrdersFile* path will override the default and will apply to every strategy in the script that does not provide its own path.

A strategy-level *OrdersFile* path is only allowed when **OrdersMode** is *Template*. In that case it writes that strategy's own order list to the given file and overrides the settings-level path for that strategy.

See **File Path Syntax** for helpful tips including special path expansion variables.

The path is expanded when the orders are generated (so it can use ?orderdate?). If it then contains an unrecognized ?token? or the file cannot be created, the run reports the problem and, in **command line mode**, exits with code 9 rather than logging success with no orders written.

See **OrdersTemplate** and **CSV Order Baskets** for details about CSV order lists.

14.20.356. OrdersFreeCash

Category

Settings or **Strategy Elements** or **Order Settings**

Description

Path and name of a text file that contains the current Free Cash Value (cash available to invest or withdraw) of your brokerage account

Notes

This setting can be used to ensure that **orders are generated** using the correct **MinFreeCash** constraint for your actual account.

See also **OrdersNetLiq**.

14.20.357. OrdersIgnoreSplits

Category

Strategy Elements

Description

Specifies that split factors in this strategy's data represent dividend adjustments rather than share count changes and should be ignored when generating orders

Choices

False - orders use normal split adjustment (default)

True - order quantities are never split-adjusted

Notes

Data imported with a dividend-adjusted close and no separate dividend amounts (for example a CSV import with an *AdjClose* field, as is typical for European stock data) causes each dividend to appear as a small price adjustment factor. RealTest cannot distinguish these from actual splits, so order generation would otherwise scale exit order quantities by them and report them in the splits CSV, creating discrepancies with the positions actually held at the broker.

When *OrdersIgnoreSplits: True* is specified, exit order quantities always match the position as entered, the strategy's positions are never included in the splits CSV that accompanies CSV order lists, and imported **TradeList** prices and quantities are used as-is.

This includes transaction-style trade lists (those with an *Action* column): buys and sells are matched by their as-traded quantities, so a sale after an adjustment date fully closes the earlier lots it came from. In a backtest, open positions are still adjusted by these factors from day to day so that their value stays continuous. When an imported trade exits, its stated exit quantity is used and the value of the adjustment is credited to the trade as a dividend.

Order prices are not affected, since a dividend does reduce the actual traded price on its ex-date.

If an ignored adjustment ratio differs from 1.0 by more than 5%, a warning is included in the generated orders output, since a ratio that large may indicate an actual split that will require manual position adjustments.

See also **OrdersUseQtyIn**, a Settings option which applies only to exit order quantities and affects all strategies.

14.20.358. 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.

14.20.359. OrdersLiveData

Category

Settings or **Order Settings**

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**.

14.20.360. OrdersMktAsLmtPct

Category

Settings or **Strategy Elements** or **Order Settings**

Description

Allows generated market orders to optionally be converted to limit orders X% beyond the last close.

Input

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 is 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 *OrdersMktAsLmtPct* 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 Close * 1.1 as its limit price, and a long exit (sell) order would use 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:

OrdersMktAsLmtPct: 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.

14.20.361. OrdersMode

Category

Settings or **Order Settings**

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.

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**

14.20.362. OrdersNetLiq

Category

Settings or **Strategy Elements** or **Order Settings**

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 2022 with an account balance of $100K.

Your Settings might look like this:

```
Settings:
   AccountSize: 100000
   StartDate: 2022-01-01
   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
QtyType: Shares
```

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.

See also **OrdersFreeCash**.

14.20.363. OrdersTemplate

Category

Settings or **Strategy Elements** or **Order Settings**

Description

Path and name of the CSV order list template file to use for generating this test's order basket

Notes

An *OrdersTemplate* path must be specified when **OrdersMode: Template** is used and can optionally be specified for **OrdersMode: OrderClerk**.

A settings-level *OrdersTemplate* path will apply to every strategy in the script that does not provide its own path.

A strategy-level *OrdersTemplate* path will apply only to the strategy that specifies it.

See **OrdersFile** and **CSV Order Baskets** for details about CSV order lists.

14.20.364. OrdersUseQtyIn

Category

Settings or **Order Settings**

Description

Forces generated exit order quantities to match the position's entry quantity

Choices

True - change exit quantity to match entry quantity if needed (ignore any splits between them)

False - (default) adjust exit quantity for any splits that happened since entry

Notes

This is an advanced option provided for users who want to use **Adjustment: TotalReturn** data in their strategies.

Total-return adjustment literally means convert dividends to splits. Once data is imported that way, dividends become indistinguishable from normal stock splits (which are more rare).

A normal stock split changes the number of shares held in any open positions on the ex-split date. RealTest by default accounts for this fact when it generates exit orders (also in the trade lists of backtests).

It is generally recommended to avoid using total-return adjustment for any live strategies because it can make exit order quantities incorrect, since the actual corporate action was a dividend, not a split.

This option is provided to support users who nevertheless prefer total-return adjustment.

If you use this option, be sure to keep an eye out for real splits in your positions. You will need to manually correct their exit order sizes.

See also **OrdersIgnoreSplits**, a per-strategy version of this idea which additionally excludes the strategy's positions from the splits CSV and warns when a large ratio is ignored.

14.20.365. OrdersUseTLEndDate

Category

Strategy Elements

Description

Specifies whether **TLEndDate** is applied in Orders mode for this strategy

Choices

False - ignore TLEndDate in Orders mode (default)

True - use TLEndDate in Orders mode

Notes

By default, a strategy defined with both a **TradeList** and strategy **rule formulas** will play back the trade list to its end and then use the strategy rules to generate orders. This assumes that the trades in the trade list correctly define the current positions in the live account. TLEndDate, if present, is therefore ignored in Orders mode.

For situations when it's not practical to maintain a live trade list daily (e.g. no automated connection to a broker, too much work to do the data entry, etc.), *OrdersUseTLEndDate: True* allows the strategy rules to act as a proxy for the missing live trade records after the specified TLEndDate.

14.20.366. OrderSum

Category

General-Purpose Functions

Description

Evaluates an expression for each non-skipped setup and returns the sum of the values

Syntax

OrderSum(expression, symbol {0})

Notes

This is a specialized function meant to be used in entry-related formulas that want 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.

Scope: in a strategy context the sum covers that strategy's own setups. Inside *Combined(...)* or *Extern(@group, ...)* it covers the setups of all member strategies. Timing follows the same rules as **IsOrder**: sums over strategies in an earlier **EntryPhase** are final, while sums over strategies in the current phase include only the selection turns completed so far.

The optional *symbol* argument (a symbol number, default 0 = all symbols) restricts the sum to setups for that one symbol.

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.

14.20.367. 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 within the existing date range of each stock, so no bars are added before the start of its data. The most recent missing bars are, however, filled forward to the latest import date (bounded by the PadTrailMaxDays setting), so bars can be added after the end of a symbol's own 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.

Like **Padding**, this can be specified per **DataPath** rather than only at the Import-block level, so each DataPath can align to its own reference symbol.

14.20.368. 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.

Padding (and the corresponding **PadAlignSym**) can be specified per **DataPath** rather than only at the Import-block level. A path-level value overrides the block-level default for symbols read from that path. This makes it possible, for example, to align each country's stocks to that country's local index symbol (US stocks to $SPX, AU stocks to $XAO.au, etc.) within a single multi-country import.

Padding also fills a symbol's most recent missing bars. If a symbol's last bar is older than the newest date in the import -- for example, its market was closed for a holiday that other markets observed -- padding bars are added forward to that latest date so the symbol stays aligned with the global calendar, which keeps a later backtest consistent with the orders that were generated on those dates. The trailing fill is bounded by the *PadTrailMaxDays* setting in RealTest.ini (default 10 trading days; 0 disables it) so a feed that has simply stopped is not carried forward indefinitely. Delisted and expired symbols are never trailing-padded.

See also the general topic of **Calendar Alignment**.

14.20.369. Parameters

Category

Script Sections

Description

Optimization Parameter Definitions

Notes

See **Parameters Section** and **Multiple Tests and Optimization**.

14.20.370. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Examples

```
PlusDI: PDI(14)
MinusDI: MDI(14)
EntrySetup: PlusDI > MinusDI
```

Enter when the positive directional index exceeds the negative.

14.20.371. 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

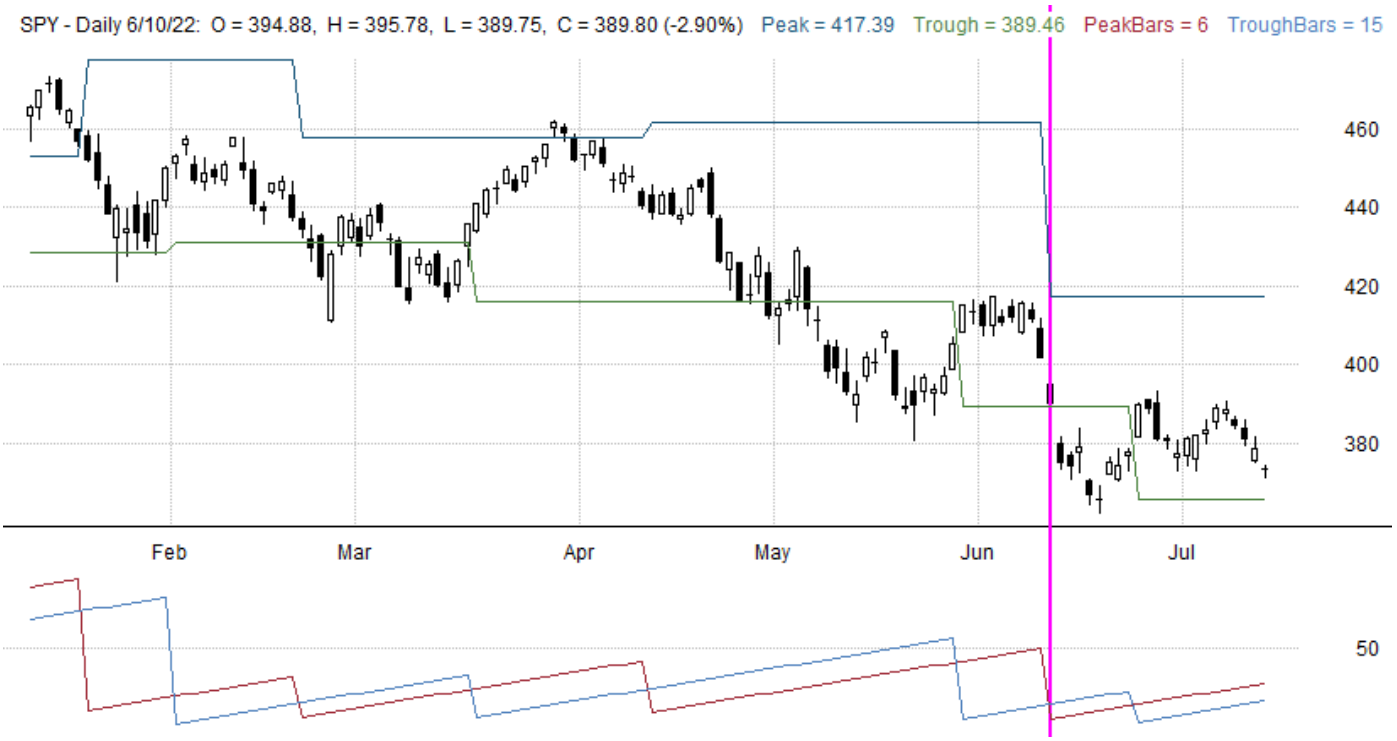
least n% below that most recent high.

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.



Example

```
LastPeak: Peak(H, 10)
```

Most recent high followed by a 10% drop. Does not look ahead.

14.20.372. 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.

Example

```
BarsSincePeak: PeakBars(H, 10)
```

Bars since the most recent peak (defined by a 10% drop from the high).

14.20.373. 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 / (count-1) * (count-rank), or simply 1 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).

Example

```
PRank: PercentRank(C, 252)
```

Where today's close ranks in the past year (100 = highest, 0 = lowest).

14.20.374. 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.

Example

```
Q25: PercentRankN(25, C, 252)
```

The 25th percentile close price over the past year.

14.20.375. 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.

14.20.376. PrevExitLimit

Category

Current Position Information

Description

Previous exit limit (target) price

Notes

At position entry time, *PrevExitLimit* is initialized to a very high value for a long position, or a very low value for a short position, so that on the first bar a `Min()` (long) or `Max()` (short) formula picks up the newly-computed target.

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).

Example

```
ExitLimit: Min(PrevExitLimit, C + 3 * ATR(14))
```

Trailing target: today's target is the lower of yesterday's target or a new ATR-based level.

14.20.377. PrevExitStop

Category

Current Position Information

Description

Previous exit stop price

Notes

At position entry time, *PrevExitStop* is initialized so that it will not limit the first bar's stop: a large negative value for a long position (so `Max()` returns the new **ExitStop** level) or a large positive value for a short position (so `Min()` returns the new level).

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)
```

Example

```
ExitStop: Max(PrevExitStop, C - 2 * ATR(14))
```

Trailing stop: today's stop is the higher of yesterday's stop or a new ATR-based level.

14.20.378. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
CumRet: Product(1 + ROC(C, 1) / 100, 20) - 1
```

Compound return over 20 bars, equivalent to ROC(C, 20) / 100.

14.20.379. PositionSum

Category

General-Purpose Functions

Description

Evaluates an expression for each currently open position and returns the sum of the values

Syntax

PositionSum(expression, symbol {0})

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.

The optional *symbol* argument (a symbol number, default 0 = all symbols) restricts the sum to open positions for that one symbol.

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 generation. 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.

Example

```
TotalExp: PositionSum(Shares * C)
```

Sum of an expression evaluated for each open position in the current strategy (and any strategies grouped with it).

14.20.380. 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.

14.20.381. QtyFinal

Category

Strategy Elements

Description

Can be used to modify entry position sizes after top-down setup selection has been completed

Input

Any formula specifying a number of shares or contracts, or a position value, or an allocation percentage

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**

14.20.382. 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)

FillPrice - use the entry price

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.

14.20.383. 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).

Positions imported from a **TradeList** (including the OrderClerk trade list) are rounded only when QtyRound is explicitly specified, so that trade lists for fractional-quantity instruments (crypto, forex) play back exactly as recorded. If such a position holds a fractional quantity and you want its **DynamicSizing** rebalance and exit orders in whole shares, add *QtyRound: 1* to the strategy.

14.20.384. 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 the order price (or the fill price if QtyPrice: FillPrice is set)

Percent - *Quantity* represents a percentage of current allocation (**S.Alloc**)

Notes

During setup selection, **Quantity** is calculated and constraints (such as **MaxInvested**) are applied based on the **OrderPrice** of the **EntrySetup**.

To "look ahead" to the actual entry price when calculating position sizes, specify **QtyPrice**: FillPrice.

The following are all equal position sizes:

```
▽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
```

When using **Currency Conversion**, note that quantities calculated with **QtyType: Shares** are NOT converted. Quantities expressed in Shares (or Contracts) are interpreted as verbatim share (contract) counts. Divide your Quantity formula by **FxRatio** if you want your shares-type quantities converted. Quantities calculated as Value or Percent, on the other hand, are always FX-converted automatically.

14.20.385. 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.

See **DynamicSizing** for information on how *Quantity* can be used differently in this special alternative strategy mode.

14.20.386. 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.

If called with one parameter, *Random(n)* returns a random integer between 0 and *n* inclusive. If called with two parameters, *Random(a, b)* returns a random integer between *a* and *b* inclusive. (In both cases the step defaults to 1, giving integer results; supply all three parameters for decimal results rounded to step.)

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)*.

Example

```
RandVal: Random(1, 100, 1)
```

Random integer from 1 to 100. With no parameters, returns a decimal between 0 and 1.

14.20.387. RandomSeed

Category

Settings

Description

Permits use of the same sequence of pseudo-random numbers every time a script is run; a number, or an expression of Parameters evaluated for each test

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 seed is passed to the C **srand** function and the **rand** function is then used for each random value needed. Each test seeds its own sequence after its Data items have been calculated (Data items are calculated by several threads, each seeded from *RandomSeed* plus its thread number, so a Data item that uses **Random** gets the same values in every test unless the seed itself changes).

A constant *RandomSeed* is offset by each test's position in an **Optimization** (0 for the first test), so the tests draw different random values from each other while the run as a whole reproduces exactly when repeated. A single test uses the seed itself.

A *RandomSeed* that references **Parameters** is evaluated for each test and used exactly as evaluated, so any test of an optimization can be reproduced on its own by running a single test with that test's parameter values. For example, with a parameter *pass* that is not used anywhere else, *RandomSeed: 1000 + pass * 977* gives each value of *pass* its own sequence, which a single test with that *pass* value repeats. This is a convenient way to run a strategy with *SetupScore: Random()*

many times to see the range of results that random selection among tied setups can produce.

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.

14.20.388. 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**.

14.20.389. 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**.

Example

```
CRank: Rank(C, 20)
```

Rank 1 = highest value; rank 20 = lowest (for count=20).

14.20.390. 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.

Example

```
SecondHigh: RankN(2, H, 20)
```

The second-highest High value in the past 20 bars.

14.20.391. Reason

Category

General-Purpose Functions

Description

Returns both a value and a string

Syntax

Reason(value, string)

Parameters

value - numeric return value

string - string return value

Notes

This special-purpose function serves to allow a single **ExitLimit** or **ExitStop** formula to return different prices under different conditions and cause the resulting trade record to include a description of the condition that triggered the exit in its **T.Reason** field.

This capability was previously supported only in **ExitRule** by using **Select**. That worked implicitly because *ExitRule* is a true/false condition. If the formula returned any string value, that was interpreted as "true".

In contrast, *ExitLimit* and *ExitStop* need to return a specific price, so a function that can return both a value and a string was required to enable reason strings.

Example

```
ExitStop: if(Extern($SPY, C > MA(C, 100)),
   Reason(FillPrice * 0.6, "bull market stop"),
   Reason(FillPrice * 0.9, "bear market stop"))
```

14.20.392. Reduce

Category

Strategy Elements

Description

Specifies whether to reduce quantity when a new position's size would push total investment above a **capacity constraint**

Input

0 or false - don't reduce (default)

value or true - reduce to available capacity (but not less than this dollar size value) if necessary

Notes

If a non-zero value is specified, a new position that would exceed a capacity constraint is reduced to the size of the remaining available capacity, provided it is not less than the dollar value specified. Otherwise the entry is skipped.

When "true" is specified, this is internally represented as 1, so the minimum position size is $1.

14.20.393. RemoveLatest

Category

Import Specification

Description

Whether to remove the global latest bar from any symbol that has it, or remove only in-progress session bars

Choices

False - don't remove the latest bar (default)

True - remove the latest bar

Partial - remove a symbol's latest bar only if its exchange session had not yet completed at import time (Yahoo imports only)

Notes

When this import setting is true, the latest bar is removed from any symbol that had the global latest (latest latest) bar.

No bars are removed from symbols that do not have the global latest bar.

This option can be used either to remove the latest bar from all symbols when they all have the same latest bar, or to equalize the latest bar when some symbols have an extra bar (e.g. in a mixed US and ASX import done between the ASX and US closes).

While an exchange's session is in progress (which for some exchanges includes the pre-open auction period), Yahoo serves an in-progress bar for the current date. An import done at that time ends on that partial bar, which shifts generated orders forward a day: entry setups are evaluated against the thin partial bar (and typically fail) while exit orders still appear.

RemoveLatest: Partial addresses this directly. Each symbol's latest bar is removed only when that symbol's own exchange session had not yet completed at import time, using the session schedule metadata Yahoo provides with each symbol's data. Because session completeness is judged per symbol in exchange time, a mixed-market import does the right thing regardless of when it runs: an in-progress bar is removed while a completed bar from another market is kept, even when both carry the same calendar date. Unlike True, Partial never removes a completed bar, so it is safe to leave enabled in an import used for pre-open order generation.

Partial applies to Yahoo imports only (other daily data sources do not provide session schedule metadata). Any removed bars are listed in the import log when **LogFile** is specified.

14.20.394. RepairData

Category

Import Specification

Description

Whether to detect and repair common data errors during import

Choices

False - don't repair data (default)

True - repair data

Notes

When enabled, RealTest scans all imported bars after loading and attempts to fix the following types of data errors:

1. Zero OHLC fields - a bar with a zero Open, High, Low, or Close (but at least one positive field) has the zero field filled in from the previous close or the bar's other OHLC fields.
2. 100x field errors - a single OHLC field that is approximately 100x or 1/100x the median of the other three fields is divided or multiplied by 100.
3. Impossible bars - High/Low values that don't contain Open/Close are clamped to valid ranges (e.g. if Close > High, High is set to Close).
4. Flat spike bars - Bars where Open = High = Low = Close with zero volume, at a price different from the previous close, are replaced with the previous close. These typically indicate placeholder bars inserted by the data vendor. Bars that appear to be legitimate close-only data (neighboring bars also have O=H=L=C) are left unchanged.
5. 100x price errors - Bars where the split-adjusted price is approximately 100x or 1/100x the previous bar, without a corresponding split event. Both scattered errors (a short run of wrong bars bracketed by opposite transitions) and block errors (all bars before or after a single transition) are detected and corrected.
6. Duplicate dividends - Dividend events within 7 days of each other with amounts within 10% are assumed to be duplicates; the later one is removed.
7. 100x dividend errors - Dividend amounts that exceed the closing price on the ex-date are assumed to be 100x too large and are divided by 100.

All repairs are logged to the import log file (if enabled) with the symbol, date, and a description of each correction. The log also reports the total number of repairs made.

14.20.395. 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

Example

`Clean: Replace(?Name, " Inc.", "")`

Remove all occurrences of a substring.

14.20.396. ReportContent

Category

Settings

Description

Specifies which optional sections to include when a **Test Summary Report** is generated

Choices (multiple, separated by commas)

Settings - include the test settings and parameter list

Stats - include the test summary stats

Monthly - include the monthly returns table

Correl - include the strategy correlation matrix

Graphs - include the stats graphs

MonteCarlo - include Monte Carlo analysis table and plots

Plots - include trade scatter plots

Script - include the script used to run the test

Notes

When *ReportContent* is specified by a script, the choices persist until changed later, either by script or on the Reports tab of the **Program Options** dialog.

14.20.397. ReportsFolder

Category

Settings

Description

Path of the folder in which **Test Summary Reports** are created.

Notes

The default *ReportsFolder* is *(RealTest Installation Folder)\Output\Reports*.

If *Output* has been changed under *Default Folder Paths* in **Program Options** then the default is *Reports* in that folder.

Each new report is created in a separate sub-folder of the *Reports* folder.

14.20.398. Rescale

Category

General-Purpose Functions

Description

Maps a value from one scale to another

Syntax

Rescale(value, valmin, valmax, scalemin, scalemax, bound {0})

Parameters

value - the value to rescale

valmin - the bottom of the range that *value* comes from

valmax - the top of the range that *value* comes from

scalemin - the bottom of the range to scale *value* to

scalemax - the top of the range to scale value to

bound {0} - whether *value* < *valmin* or *value* > *valmax* should be adjusted before rescaling

Return Value

If bound is zero or omitted: *(value - valmin) / (valmax - valmin) * (scalemax - scalemin) + scalemin*

If bound is non-zero: *(Bound(value, valmin, valmax) - valmin) / (valmax - valmin) * (scalemax - scalemin) + scalemin*

Example

```
NormalizedTurnover: Rescale(Close * Volume, 1e5, 1e12, 1, 10, 1)
```

14.20.399. Results

Category

Script Sections

Description

Results column definitions

Notes

See **Results Section** and **Results Windows**.

14.20.400. ResultsByYear

Category

Settings

Description

Add per-calendar-year statistic columns to results windows

Syntax

ResultsByYear: *list*

where *list* is any combination of ROR, MaxDD, and Trades (comma-separated), or None

Choices

None - no per-year columns (the default)

ROR - rate of return for each calendar year

MaxDD - maximum drawdown within each calendar year

Trades - number of trades (exits) closed in each calendar year

Notes

When one or more metrics are specified, a band of columns is added to the right of the **results window**, with one column per metric per calendar year that the test spans (e.g. ROR 2018, ROR 2019, ..., MaxDD 2018, ...).

The per-year values are computed from the same combined-strategy statistics used to build the Combined Monthly Percent Gains report table, so they match that report: ROR is compounded via the year's TWEQ ratio (or non-compounded as year P&L over starting equity when compounding is off), MaxDD resets each calendar year, and Trades counts the year's exits.

These columns are recomputed from stored statistics at display time; nothing is added to the RTR file, and enabling them adds the columns to every open results window. The generic list-to-CSV export (**SaveTestListAs**) includes the per-year columns.

ResultsByYear can be set three ways, in increasing precedence: the RealTest.ini default, the **Outputs** menu "Results By Year" toggles (applied live to open windows, so columns can be added to existing or optimization output after the fact), and a script's ResultsByYear: setting.

The script setting applies to optimization results as well: the per-year band appears in the optimization results window and in the **SaveTestListAs** CSV written by a batch (-optimize) run.

14.20.401. 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.

Restating this setting with a blank value clears it. This is useful when a script uses **Include** to reference another script that specifies a ResultsFile: a blank ResultsFile after the Include prevents runs of the including script from adding rows to that results file.

See **File Path Syntax** for helpful tips including special path expansion variables.

14.20.402. 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

Example

```
Suffix: Right(?Symbol, 2)
```

Last 2 characters of the symbol name.

14.20.403. 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} SQR(S.BPY)*Avg((S.NetPct-S.RiskFreeRate/S.BPY),Periods)/StdDev(S.NetPct,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*.

Do not prefix the symbol with $ -- that is only required for symbol references within formulas.

The referenced symbol must also have been added to the **Import** of the current data file.

14.20.404. 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)*.

Example

```
Return1M: ROC(C, 21)
```

ROC and PctChg are interchangeable names. Returns 100 * (C / C[21] - 1).

14.20.405. RollCost

Category

Strategy Elements

Description

Total cost of each roll transaction while holding a futures position.

Input

Formula specifying a dollar amount.

Notes

RollCost is only applied to trades in a strategy that uses back-adjusted continuous futures data, such as **CSI Futures** or **Norgate** continuous contracts.

Each time an open position is held across a roll date boundary, the *RollCost* formula is evaluated.

The dollar amount that it returns is added to the running total of roll costs for this position.

When the trade exits, this total is subtracted from its net profit calculation (similarly to commissions).

To see the total roll costs for a trade, add a column to your Trades.rts that shows **T.RollCosts**.

To see the total daily roll costs for all trades exited that day, add an item to your Graphs.rts that shows **S.RollCosts**.

Note that *RollCost* should calculate the total commissions plus slippage impact for both transactions (or for a spread trade).

Example

If your **Commission** is $1.50 per contract and your **Slippage** averages one tick, then your formula would be:

```
RollCost: 2 * Contracts * (1.5 + (1 * TickSize * PointValue))
```

14.20.406. 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 >= 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.

- round(-1.5, 1, 0) = -2
- round(-1.5, 1, 1) = -2
- 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.)

Example

```
RndPrice: Round(C * 1.05, 0.01)
```

Round to nearest penny. Direction: 0=nearest, 1=up, -1=down.

14.20.407. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

By default, RealTest seeds Wilder's smoothing from a symbol's first bar (matching AmiBroker). Checking "Initialize RSI with SMA" on the Calculation tab of **Program Options** instead seeds from the average of the first *len* changes (matching Wilder's book, TradingView, and QuantConnect); the two methods differ only during the warm-up region, about the first 3-4x *len* bars.

Examples

```
MyRSI: RSI(2)
EntrySetup: MyRSI < 10
```

Look for extreme oversold conditions using a short-period RSI.

14.20.408. 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 **one-pass calculation** when used in the **Data Section** with a non-variable length.

Example

```
EntryLimit: RRSI(2, 10)
```

The price at which RSI(2) would equal 10 on the next bar, used here as a limit entry price.

14.20.409. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VolRSI: RsiF(V, 14)
```

Unlike RSI (which uses Close), RsiF calculates RSI for any data series.

14.20.410. S.Alloc

Category

Test Statistics Arrays

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**. Note that in a combined-compounding test this makes a strategy's percentage drawdown a measure of what its losses cost the total account, not the percent decline of the strategy's own equity line - see **Asset Allocation and Position Sizing**.

See **Asset Allocation and Position Sizing** for additional information.

14.20.411. S.BPx

Category

Test Statistics Arrays

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.BPY / 4)
- S.BPM = bars per month (S.BPY / 12)
- S.BPW = bars per week (S.BPY / 52)

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 by default.

Use the **DaysPerYear** setting to specify a number of daily stat periods per year other than 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 statistics, the values returned will be the same for every strategy regardless of strategy-specific bar size.

14.20.412. S.CashInOut

Category

Test Statistics Arrays

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]*

Unlike **S.FeesInOut**, *S.CashInOut* is NOT included in backtest stats such as the annual return, max drawdown, and daily net percent gain or loss.

14.20.413. S.Collateral / S.FuturesMargin

Category

Test Statistics Arrays

Description

The current total dollar collateral (margin) required to hold open futures, FX, and crypto positions

Notes

S.Collateral was formerly named *S.FuturesMargin*, which is still accepted as an alias.

See **Collateral** for details about how RealTest models collateral requirements, and how the presence of a *Collateral* formula changes the free cash calculation.

14.20.414. S.Comms

Category

Test Statistics Arrays

Description

The sum of all **commissions** paid by this strategy on this test date

14.20.415. S.Compounded

Category

Test Statistics Arrays

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").

14.20.416. S.Date

Category

Test Statistics Arrays

Description

The end date of the current stat period

14.20.417. S.DDBars

Category

Test Statistics Arrays

Description

The duration, in bars, of the current drawdown for this strategy as of this test date

14.20.418. S.DDDlr

Category

Test Statistics Arrays

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.

14.20.419. S.DDPct

Category

Test Statistics Arrays

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.

14.20.420. S.Dividends

Category

Test Statistics Arrays

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)*.

14.20.421. S.Entries

Category

Test Statistics Arrays

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**.

14.20.422. S.EntryOrders

Category

Test Statistics Arrays

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**.

14.20.423. S.Equity

Category

Test Statistics Arrays

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*.

Likewise for **FeesInOut** and **S.FeesInOut**, the difference being that *CashInOut* is not included in backtest stats while *FeesInOut* are included.

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)*.

14.20.424. S.Exits

Category

Test Statistics Arrays

Description

The number of positions exited by this strategy on this test date

14.20.425. S.Exposure

Category

Test Statistics Arrays

Description

Total value of currently open positions as a fraction of current allocation (typically mark-to-market account value)

Notes

S.Exposure is **S.Invested** / **S.Alloc**

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.426. S.FeesInOut

Category

Test Statistics Arrays

Description

The running total net **FeesInOut** for this strategy as of this date

Notes

The net fees in-out for one stats period can be calculated as *S.FeesInOut - S.FeesInOut[1]*

Unlike **S.CashInOut**, *S.FeesInOut* is included in all the stats calculations of a backtest, such as the annual return, max drawdown, and daily net percent gain or loss.

14.20.427. S.First

Category

Test Statistics Arrays

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 in 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 **Test 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.

14.20.428. S.FreeCash

Category

Test Statistics Arrays

Description

The current free cash balance of the account

Notes

S.FreeCash is calculated as:

(starting equity)

+ (closed trade net profit incl. dividends)

+ (net cash in-out)

+ (net interest received-paid)

+ (dividends accrued to currently open positions, only when **DividendsToCash** is True)

+ (proceeds from currently open short stock positions)

- (cost basis of currently open long stock positions)

- (collateral required for currently open futures, FX, and crypto positions)

Combined(S.FreeCash) represents the current "free cash balance" of the trading account.

See also: **CashIntPct**, **MarginIntPct**, **Collateral**, **MinFreeCash**, **DividendsToCash**

14.20.429. S.Interest

Category

Test Statistics Arrays

Description

The net interest received and/or paid for excess cash and/or margin loan and/or short borrow fees on this date

Notes

S.Interest is only relevant in the **Combined** stats series. It will always be 0 for individual strategies.

Cumulative net interest can be calculated as *Sum(S.Interest, S.Number)*.

See **CashIntPct**, **MarginIntPct**, and **RiskFreeRateSym** for details on how interest received and/or charged is calculated and applied.

14.20.430. S.Invested

Category

Test Statistics Arrays

Description

Total dollar value of currently open positions

Notes

S.Invested is the sum of (shares * latest price) of open positions when **MarkToMarket** is True (the default).

When *MarkToMarket* is False *S.Invested* is the sum of (shares * entry price) of open positions.

To recover this original (cost-basis) investment value when *MarkToMarket* is True, use *S.Invested - S.M2M* for a long-only strategy. If short positions are also open, add twice the short mark-to-market: *S.Invested - S.M2M + 2 * S.ShortM2M*.

For futures positions "shares" above means (contracts * point value), i.e., *S.Invested* uses notional dollars.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.431. S.LongExp

Category

Test Statistics Arrays

Description

Total value of currently open long positions as a fraction of current allocation (typically mark-to-market account value)

Notes

S.LongExp is **S.LongInv** / **S.Alloc**

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.432. S.LongInv

Category

Test Statistics Arrays

Description

Total dollar value of currently open long positions

Notes

S.LongInv is the sum of (shares * latest price) of open long positions when **MarkToMarket** is True (the default).

When *MarkToMarket* is False *S.LongInv* is the sum of (shares * entry price) of open long positions.

To recover this original (cost-basis) long investment value when *MarkToMarket* is True, use *S.LongInv - S.M2M* for a long-only strategy. If short positions are also open, add the short mark-to-market: *S.LongInv - S.M2M + S.ShortM2M*.

For futures positions "shares" above means (contracts * point value), i.e., *S.LongInv* is the notional dollar value of open longs.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.433. S.LongM2M

Category

Test Statistics Arrays

Description

Net gain or loss, in dollars, of the currently open long positions in this strategy, measured from their entry prices

Notes

S.LongM2M is the long-side portion of **S.M2M**; the short-side portion is **S.ShortM2M**, and S.M2M = S.LongM2M + S.ShortM2M.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.434. S.LossAmt

Category

Test Statistics Arrays

Description

The sum of dollar (or base currency) losses for all trades exited as losses by this strategy on this test date (expressed as a positive value).

Can also be written as *S.LossDlr*.

14.20.435. S.LossBars

Category

Test Statistics Arrays

Description

The sum of durations, in bars, for trades exited as losses by this strategy on this test date

14.20.436. S.Losses

Category

Test Statistics Arrays

Description

The number of trades exited as losses by this strategy on this test date

14.20.437. S.LossPct

Category

Test Statistics Arrays

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*.

14.20.438. S.LossPctAlloc

Category

Test Statistics Arrays

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 of *dollar loss / dollar position size.*

14.20.439. S.M2M

Category

Test Statistics Arrays

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.

14.20.440. S.MAE

Category

Test Statistics Arrays

Description

Worst-case ("adverse") net gain or loss, in dollars, of all positions remaining open in this strategy on this test date

Notes

Each open position contributes its adverse excursion from its entry price: the (negative) amount by which it would have been under water had it been marked at its least favorable price of the day. The excursion is floored at zero per position, so a position whose least favorable price of the day is still above its entry price contributes nothing, no matter how far it fell during the day. *S.MAE* is the sum of those per-position values - a portfolio-level total of conventional per-trade MAE.

For the worst theoretical portfolio-level P&L of the day relative to the prior close (no entry prices, no per-position floor, gaps included), see **S.NetDlrMin**.

The default "MAE" **daily stats graph** is calculated as *S.MAE* / **S.Alloc** and displayed as a percentage.

14.20.441. S.MFE

Category

Test Statistics Arrays

Description

Best-case ("favorable") net gain or loss, in dollars, of all positions remaining open in this strategy on this test date

Notes

Each open position contributes its favorable excursion from its entry price: the (positive) amount by which it would have been in profit had it been marked at its most favorable price of the day. The excursion is floored at zero per position, so a position whose most favorable price of the day is still below its entry price contributes nothing. *S.MFE* is the sum of those per-position values - a portfolio-level total of conventional per-trade MFE.

For the best theoretical portfolio-level P&L of the day relative to the prior close (no entry prices, no per-position floor, gaps included), see **S.NetDlrMax**.

The default "MFE" **daily stats graph** is calculated as *S.MFE* / **S.Alloc** and displayed as a percentage.

14.20.442. S.MarkedToMarket

Category

Test Statistics Arrays

Description

Strategy mark-to-market flag

Notes

Returns 1 (true) if this strategy marks its open positions to market (the default), or 0 (false) otherwise.

A strategy marks to market unless its **MarkToMarket** setting is False, in which case equity and invested values change only when positions are closed.

This is the mark-to-market counterpart of **S.Compounded**.

14.20.443. S.MaxAlloc

Category

Test Statistics Arrays

Description

The amount, in dollars, of the largest allocation value of this strategy as of this test date.

14.20.444. S.MaxDDBars

Category

Test Statistics Arrays

Description

The duration, in bars, of the longest drawdown as of this test date

Notes

The longest drawdown is not necessarily the largest one.

14.20.445. S.MaxDDDlr

Category

Test Statistics Arrays

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.

14.20.446. S.MaxDDPct

Category

Test Statistics Arrays

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.

14.20.447. S.MaxEquity

Category

Test Statistics Arrays

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.

14.20.448. S.MinAlloc

Category

Test Statistics Arrays

Description

The amount, in dollars, of the smallest allocation value of this strategy as of this test date.

14.20.449. S.MinEquity

Category

Test Statistics Arrays

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.

14.20.450. S.NetDlr

Category

Test Statistics Arrays

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 matches *S.Equity - S.Equity[1]* on a period with no cash in or out. On a period with a deposit or withdrawal, *S.NetDlr* excludes that cash movement (see **S.CashInOut**) while *S.Equity* includes it.

See also **S.NetDlrMin** and **S.NetDlrMax** - the worst- and best-case values this stat could have reached, with open positions marked at their least/most favorable prices of the period.

14.20.451. S.NetDlrMax

Category

Test Statistics Arrays

Description

The best-case value **S.NetDlr** could have reached this period, with each open position marked at its most favorable price of the period (high for longs, low for shorts) instead of its closing price. Also available as *S.NetAmtMax*.

Notes

S.NetDlrMax - S.NetDlr equals the sum over open positions of quantity × (high - close) for longs plus quantity × (close - low) for shorts. It is therefore the best theoretical mark-to-market print of the period relative to the prior close, gaps included. Entry prices are not involved; compare **S.MFE**, which measures each position's excursion from its entry price.

Positions that exited at the close of the period are still included. On a position's entry day, only excursion that could have occurred after the entry is counted; a position entered at the close contributes nothing. For a strategy with **MarkToMarket***: False*, open positions are not marked and S.NetDlrMax equals S.NetDlr.

See also **S.NetPctMax** and **S.NetDlrMin**.

14.20.452. S.NetDlrMin

Category

Test Statistics Arrays

Description

The worst-case value **S.NetDlr** could have reached this period, with each open position marked at its least favorable price of the period (low for longs, high for shorts) instead of its closing price. Also available as *S.NetAmtMin*.

Notes

S.NetDlrMin - S.NetDlr equals the sum over open positions of quantity × (low - close) for longs plus quantity × (close - high) for shorts. It is therefore the worst theoretical mark-to-market print of the period relative to the prior close, gaps included - the largest one-period loss the account could have shown had it been marked at the least favorable moment. Entry prices are not involved; compare **S.MAE**, which measures each position's excursion from its entry price.

Positions that exited at the close of the period are still included. On a position's entry day, only excursion that could have occurred after the entry is counted; a position entered at the close contributes nothing. For a strategy with **MarkToMarket***: False*, open positions are not marked and S.NetDlrMin equals S.NetDlr.

See also **S.NetPctMin** and **S.NetDlrMax**.

14.20.453. S.NetFx

Category

Test Statistics Arrays

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.

14.20.454. S.NetPct

Category

Test Statistics Arrays

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 matches *(S.Equity - S.Equity[1]) / S.Alloc[1]* on a period with no cash in or out. On a period with a deposit or withdrawal, *S.NetPct* excludes that cash movement (see **S.CashInOut**) while *S.Equity* includes it.

See also **S.NetPctMin** and **S.NetPctMax** - the worst- and best-case values this stat could have reached, with open positions marked at their least/most favorable prices of the period.

14.20.455. S.NetPctMax

Category

Test Statistics Arrays

Description

The best-case value **S.NetPct** could have reached this period, with each open position marked at its most favorable price of the period instead of its closing price

Notes

This matches **S.NetDlrMax** / *S.Alloc[1]* - the same denominator as *S.NetPct*. See **S.NetDlrMax** for how the value is derived.

14.20.456. S.NetPctMin

Category

Test Statistics Arrays

Description

The worst-case value **S.NetPct** could have reached this period, with each open position marked at its least favorable price of the period instead of its closing price

Notes

This matches **S.NetDlrMin** / *S.Alloc[1]* - the same denominator as *S.NetPct*. See **S.NetDlrMin** for how the value is derived.

14.20.457. S.Number

Category

Test Statistics Arrays

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 by default be 252 stat periods for both strategies, and *S.Number* will be 252 for the final period of either strategy.

Use the **DaysPerYear** setting to specify a number of daily stat periods per year other than 252.

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").

14.20.458. S.Positions

Category

Test Statistics Arrays

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 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**.

14.20.459. S.RiskFreeRate

Category

Test Statistics Arrays

Description

The daily closing value of the data series specified by the **RiskFreeRateSym** symbol, divided by

100 so that it is a decimal fraction rather than a raw percentage (for example, a close of 5, meaning 5%, gives 0.05)

Notes

This is mainly useful when calculating the Sharpe Ratio, as in the default formula from Results.rts:

```
Sharpe:        {#2} SQR(S.BPY)*Avg((S.NetPct-S.RiskFreeRate/S.BPY),Periods)/StdDev(S.NetPct,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.

14.20.460. S.RollCosts

Category

Test Statistics Arrays

Description

The sum of all **roll costs** paid by this strategy on this test date

14.20.461. S.Setups

Category

Test Statistics Arrays

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**.

14.20.462. S.ShortExp

Category

Test Statistics Arrays

Description

Total value of currently open short positions as a fraction of current allocation (typically mark-to-market account value)

Notes

S.ShortExp is **S.ShortInv** / **S.Alloc**

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.463. S.ShortInv

Category

Test Statistics Arrays

Description

Total dollar value of currently open short positions

Notes

S.ShortInv is the sum of (shares * latest price) of open short positions when **MarkToMarket** is True (the default).

When *MarkToMarket* is False *S.ShortInv* is the sum of (shares * entry price) of open short positions.

To recover this original (cost-basis) short investment value when *MarkToMarket* is True, use *S.ShortInv* + *S.ShortM2M*.

For futures positions "shares" above means (contracts * point value), i.e., *S.ShortInv* is the notional dollar value of open shorts.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.464. S.ShortM2M

Category

Test Statistics Arrays

Description

Net gain or loss, in dollars, of the currently open short positions in this strategy, measured from their entry prices

Notes

S.ShortM2M is the short-side portion of **S.M2M**; the long-side portion is **S.LongM2M**, and S.M2M = S.LongM2M + S.ShortM2M.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.465. S.Slips

Category

Test Statistics Arrays

Description

The sum of all **slippage amounts** applied to trade entries or exits by this strategy on this test date

14.20.466. S.StartEquity

Category

Test Statistics Arrays

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.

14.20.467. S.Stops

Category

Test Statistics Arrays

Description

The number of trades that exited because of their **ExitStop** for this strategy on this test date

14.20.468. S.Targets

Category

Test Statistics Arrays

Description

The number of trades that exited because of their **ExitLimit** for this strategy on this test date

14.20.469. S.TradeBars

Category

Test Statistics Arrays

Description

The sum of durations, in bars, for all trades exited by this strategy on this test date

14.20.470. S.TradeDlr

Category

Test Statistics Arrays

Description

The sum of dollar gains-losses for all trades exited in this strategy on this test date

14.20.471. S.TradePct

Category

Test Statistics Arrays

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*.

14.20.472. S.TradePctAlloc

Category

Test Statistics Arrays

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*.

14.20.473. S.TWEQ

Category

Test Statistics Arrays

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 at $1 and then is multiplied each day by (1 + 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**.

The compounded series shows a strategy's performance in its own terms - the growth of $1 riding its daily returns - which makes it the right graph to look at when a strategy's dollar equity line in a combined test is distorted by combined-account position sizing (see **Asset Allocation and Position Sizing**).

14.20.474. S.Usage

Category

Test Statistics Arrays

Description

Total value of positions held any time this period as a fraction of current allocation (typically mark-to-market account value)

Notes

S.Usage counts every position held at any time during the period (including positions entered and exited on the same bar), so it is greater than or equal to **S.Exposure**, which counts only positions still open at the end of the bar.

Use **Combined** or **Extern** to obtain this value for all strategies or for a specific other strategy or **StatsGroup**.

14.20.475. S.WinAmt

Category

Test Statistics Arrays

Description

The sum of dollar (or base currency) gains for all trades exited as wins in this strategy on this test date.

Can also be written as *S.WinDlr*.

14.20.476. S.WinBars

Category

Test Statistics Arrays

Description

The sum of durations, in bars, for all trades exited as wins in this strategy on this test date

14.20.477. S.WinPct

Category

Test Statistics Arrays

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*.

14.20.478. S.WinPctAlloc

Category

Test Statistics Arrays

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*.

14.20.479. S.Wins

Category

Test Statistics Arrays

Description

The number of trades exited as wins by this strategy on this test date

14.20.480. 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**.

Examples

`ExitStop: SAR()`

Uses default parameters (accel=0.02, max=0.2, len=100).

`ExitStop: SAR(0.01, 0.1)`

Custom acceleration and maximum for a tighter trailing stop.

14.20.481. 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.

Example

```
CloseSAR: SarF(C)
```

Unlike SAR (which uses highs and lows), SarF uses a single data series.

14.20.482. 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.

14.20.483. SaveLogAs

Category

Settings

Description

Path and name of a text file in which to save all log output produced during a run

Notes

If *SaveLogAs* is specified, everything that would appear in the **Log window** while the script runs is also written to the specified file. This works in any run mode, including **command line mode** runs where no log window exists.

The file is created empty when the run starts and is complete when the run ends, so it always holds the output of exactly one run. A new run of the same script overwrites it without confirmation. When an optimization is run in multiple shards, each shard child writes its own copy of the file with the shard number appended to the name, as with **ResultsFile**.

"One run" means one run mode, not one invocation. A command line that names more than one mode, such as *realtest.exe -import -test script.rts*, performs a separate run per mode, and each writes its own *SaveLogAs* file. Anything consuming these files has to read all of them, or it will miss the import phase - which is where the import warnings are. Give the name a *?time?* or *?date?* variable so the runs don't overwrite one another. Note that *?time?* resolves to one-second resolution, so two run modes that finish within the same second still produce the same file name and the later one overwrites the earlier.

When **TestOutput***: Log* is enabled for a test, the per-trade detail continues to go to that test's own TEST_nnnn.TXT file and is not duplicated; the *SaveLogAs* file receives only what the Log window itself shows.

Trade list import warnings (symbol not found, delisted symbol replaced, dates that fall on non-trading days, out-of-range trades) are logged whenever *SaveLogAs* is specified. Previously these warnings appeared only with *TestOutput: Log* or during order runs, so this setting is an easy way for automated order workflows to detect symbol changes or unusable trade list rows.

See **Symbol Changes in Open Positions** for what each of the symbol-related warnings means and what to do about it.

Log output produced between runs (for example Data > Log Data Info) goes to the Log window only.

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 Syntax** for helpful tips including special path expansion variables.

14.20.484. 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 Syntax** for helpful tips including special path expansion variables.

14.20.485. SaveScanAs

Category

Settings or **Scan 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 Syntax** for helpful tips including special path expansion variables.

14.20.486. SaveScanChartsTo / SaveTradeChartsTo

Category

Settings or **Scan Settings** or **Order Settings**

Description

Paths of folders in which to automatically save a chart image for every row of a scan or TestScan list (SaveScanChartsTo) or for every trade of a test or orders run (SaveTradeChartsTo)

Notes

If *SaveScanChartsTo* is present, then every time a scan is run, a chart image (PNG file) is saved for every item (row) of the scan. In a test or orders run it applies instead to the **TestScan** list, when the script has one and the Scan checkbox is on. If *SaveTradeChartsTo* is present, then every time a test or orders run completes, a chart is saved for every trade in its trade list. A script can use either or both; each names its own folder. Specify them in the general **Settings** section or in a **ScanSettings** or **TestSettings** or **OrderSettings** section. Optimize runs ignore them.

Before 2.0.32.8 a single *SaveChartsTo* setting (with *SaveChartsMax*) served both purposes, which meant a test or orders run with a TestScan section saved both lists' charts whenever either was wanted. The old names are no longer recognized; scripts that use them must be changed to the new names.

In an orders run (any **OrdersMode**, including OrderClerkStats playback of live trades) the trade list is the backtest's, so with **EndOfTestExits**: False the trade charts include one for each still-open position, showing its entry marker only.

Scan charts are named *symbol_date.PNG* (e.g. MSFT_20210518.PNG). Trade charts are named *symbol_entrydate_tradenumber.PNG* (e.g. MSFT_20210518_37.PNG); the trade number suffix keeps two trades that entered the same symbol on the same date from overwriting each other. Note the extension is written in upper case, which matters to anything that collects the files case-sensitively.

Existing folder contents are not deleted before new charts are added, but existing chart files with the same names will be overwritten without asking.

For a test, charts are saved for the trade list as displayed: a *Filter:* item in the **Trades Section** limits which trades get charts (e.g. *Filter: T.Profit < 0* for losing trades only), and its *Sort:* item controls the order in which they are saved. A full backtest can produce thousands of trades, so filtering (or the cap below) is recommended.

When the charts have been saved, a brief popup shows how many were written. A message box appears only if one or more chart images could not be written (an invalid or unwritable folder, for example).

SaveScanChartsMax / SaveTradeChartsMax

The companion settings *SaveScanChartsMax: N* and *SaveTradeChartsMax: N* cap the number of charts saved from the corresponding list (default is no limit). The cap is applied after the list's filter and sort, so sorting the trade list by profit ascending combined with *SaveTradeChartsMax: 50* saves charts of the 50 worst trades.

Remember that *Sort:* names a column of the **Trades Section**, so that column has to be defined in the same script. Interactively the default Trades.rts supplies one named *Profit*, but a script run from the command line inherits nothing, so it must declare its own:

```
Trades:
   Profit: {$-2} T.Profit
   Sort: Profit
```

Without the column definition the script fails to parse with "unrecognized item name in Sort definition".

Chart Appearance

Saved charts employ the current chart options - visible bar count, bar style, entry-to-exit lines, and the other chart window display settings. The recommended way to control these from a script is the **Chart Settings** keywords (ChartVisibleBars, ChartEntryExitLine, ChartShowOtherTrades, ChartTradeLabels, etc.), which also include ChartImageWidth and ChartImageHeight for the saved image's pixel size. A script that sets them renders the same charts on any machine, in any run mode.

Options a script does not set fall back to their last-used values as persisted in realtest.ini from the most recently viewed chart window. *[WindowPositions] iChartW* and *iChartH* are the saved image's default width and height in pixels; if they are absent (or hold the placeholder -2147483648 that RealTest writes before any chart window has been sized), saved charts are 1600 x 1000. *[ChartWindows]* holds everything else: ChartVisibleBars, ChartExtraBars, ChartCandlestick, ChartLogScale, ChartSplitAdjust, ChartAfterExit, ChartShowIndPane / ChartIndPanePct, ChartShowVolPane / ChartVolPanePct, ChartShowCursor / ChartCursorBar, ChartBG, ChartUpBar / ChartDownBar, Indicator0Color through Indicator9Color, ChartFont / ChartFontHeight,

ChartLineWidth, EntryExitLine, ShowOtherTrades, and ChartTradeLabels.

A few options (colors, fonts, pane height percentages, cursor bar visibility) have no script keyword yet and can only be pinned via the ini. An ini written by hand for automated runs need contain only the keys it wants to pin; the rest fall back to their defaults. RealTest does not rewrite the ini during a command line run.

Charts for a scan or trade list that has already been run can also be saved on demand via the **Scan Menu** or **Trades Menu**.

This feature was originally added to aid discretionary traders in creating a "chart book" for study purposes; the trade-chart variant also supports automated workflows such as AI-assisted trade review and daily monitoring reports built from batch runs.

14.20.487. 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.

By default the CSV file will contain a separate row per date for each individual **Strategy** and **Benchmark**, each **StatsGroup**, and **Combined**.

Use **SaveStatsInclude** to optionally select which of these stats types to include.

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.

This setting is ignored when running **multiple tests** unless the *Multi Save As CSV* box is checked in the *Optimize* dialog.

See **File Path Syntax** for helpful tips including special path expansion variables.

14.20.488. SaveStatsInclude

Category

Settings

Description

Specifies which strategy types to include in the **SaveStatsAs** CSV output file

Notes

Choices are: *Strategies, Benchmarks, StatsGroups, Combined*

More than one of these can be specified, separated by commas.

If *SaveStatsInclude* is not specified then the *SaveStatsAs* CSV file will contain all of the above strategy types that are present in the script.

When *SaveStatsInclude* is specified, only the specified strategy types are included.

14.20.489. 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 Syntax** for helpful tips including special path expansion variables.

14.20.490. 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.

This setting is ignored when running **multiple tests** unless the *Multi Save As CSV* box is checked in the *Optimize* dialog.

See **File Path Syntax** for helpful tips including special path expansion variables.

14.20.491. 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.

14.20.492. Scan

Category

Script Sections

Description

Filter and column definitions for general-purpose scans

Notes

See **Scan and TestScan Sections** for details.

14.20.493. 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.

14.20.494. ScanNoDefCols

Category

Settings or **Scan 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.

14.20.495. ScanNoHeader

Category

Settings or **Scan 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.

14.20.496. ScanNoWindow

Category

Settings or **Scan 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.

14.20.497. 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.

14.20.498. 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, value1, if(cond2, value2, if(cond3, value3, NaN)))*
- *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.

Example

```
Signal: Select(RSI(2) < 10, 2, RSI(2) < 30, 1, RSI(2) > 70, -1, 0)
```

Multi-branch alternative to nested IF. Returns 0 (the default) if no conditions match.

14.20.499. 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)
```

Example

```
RSIBounce: Sequence(10, RSI(2) < 10, RSI(2) > 50)
```

True if RSI(2) dropped below 10 and then recovered above 50, in that order, within 10 bars.

14.20.500. 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**, **OptimizeSettings** and **OrderSettings** as needed to override common settings for these specific **run modes**.

See **Settings Sections** for details.

14.20.501. 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 or in **QtyFinal**.

The setup ranking mechanism can be observed by running a test with *TestOutput: Log* enabled.

14.20.502. 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**.

14.20.503. 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.

A negative return value skips the setup **and** yields the rest of the strategy's current selection turn - no further setups are considered for this strategy until its next turn. This can be useful as a guard when reacting to another strategy's selections within the same phase (available since version 2.0.28.6a).

SetupSkip is often used with **Extern** to react to another strategy's same-bar order state, e.g. *SetupSkip: Extern(@longs, Shares > 0 or IsOrder)*. Such references are deterministic only when the referenced section is in an earlier **EntryPhase** - its order state is then final. Within a shared phase, only orders granted in earlier turns of the same bar are visible, so two same-phase strategies watching each other leapfrog turn by turn rather than seeing each other's final state. Assign distinct *EntryPhase* values as the preferred way to make cross-strategy de-confliction deterministic.

See **Backtest Engine Details** for more information about how setup selection works.

14.20.504. SetupSum

Category

General-Purpose Functions

Description

Evaluates an expression for each setup and returns the sum of the values

Syntax

SetupSum(expression, symbol {0})

Notes

This is a specialized function meant to be used in entry-related formulas that want 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.

Scope: in a strategy context the sum covers that strategy's own setups. Inside *Combined(...)* or *Extern(@group, ...)* it covers the setups of all member strategies. Timing follows the same rules as **IsSetup**: values for strategies in an earlier **EntryPhase** are final, while values for strategies in the current phase reflect processing so far.

The optional *symbol* argument (a symbol number, default 0 = all symbols) restricts the sum to setups for that one symbol.

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.

14.20.505. 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).*

14.20.506. ShortLoanIntPct

Category

Settings

Description

Annualized interest rate paid for daily value of short stock positions (short borrow fee rate).

Notes

Daily short borrow fees are subtracted from combined **S.Equity** and therefore become part of the total return of the backtest.

Use **ShortPrcdsIntPct** to specify interest received for the cash value of short positions, if applicable to your broker.

14.20.507. ShortPrcdsIntPct

Category

Settings

Description

Interest rate received for value of proceeds from short stock positions held in the account.

Notes

If the settings also provide a **RiskFreeRateSym** then *ShortPrcdsIntPct* is **added** to the current daily interest rate as determined by today's value of the risk-free rate series. In this case,

ShortPrcdsIntPct 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 *ShortPrcdsIntPct* is provided when there is no *RiskFreeRateSym* then *ShortPrcdsIntPct* 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 **CashIntPct**, **MarginIntPct**, and **ShortLoanIntPct**.

14.20.508. 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.

14.20.509. 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.

14.20.510. 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.

Example

```
TrendDir: Sign(C - MA(C, 200))
```

Returns 1 above the 200-day MA, -1 below, 0 if exactly equal.

14.20.511. SigSpike

Category

Indicator Functions

Description

Adam Grimes' Sigma Spike indicator

Syntax

SigSpike(len)

Parameters

len - lookback period

Notes

Sigma Spike is defined as *ROC(C,1) / StdDev(ROC(C,1),len)[1]*.

Example

```
Spike: SigSpike(100)
```

Large negative values indicate a sharp downward price spike relative to recent volatility.

14.20.512. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
DaysFromHigh: SinceHigh(H, 252)
```

Returns 0 if today is the 252-bar high.

14.20.513. 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 low 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
DaysFromLow: SinceLow(L, 252)
```

Returns 0 if today is the 252-bar low.

14.20.514. 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 **one-pass calculation** when used in the **Data Section** without a count argument.

Example

```
DaysSinceGap: SinceTrue(O > C[1] * 1.03)
```

Returns 0 if true today, -1 if never true.

14.20.515. 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/π).

Example

```
SinVal: Sine(30)
```

Returns 0.5. Parameter is in degrees.

14.20.516. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **StdDev** and **Kurtosis**.

Example

```
Skew: Skewness(ROC(C, 1), 100)
```

14.20.517. SkipNans

Category

General-Purpose Functions

Description

Evaluate a multi-bar function using only non-NaN values within the specified lookback period

Syntax

SkipNans(expression)

Parameters

expression - any formula, typically a multi-bar function

Examples

```
SkipNans(MA(data_value, 20))
```

If the data_value array has 3 NaNs in the past 20 bars, calculate the average of the 17 non-NaN values.

```
SkipNans(Median(if(C > C[1], C/C[1]-1, nan), 100))
```

Calculate the median up-day gain in the past 100 days.

In this example we deliberately supply NaN to the Median formula on down days.

Without SkipNans this would cause the Median to be returned as NaN.

SkipNans can be used in this way to calculate any multi-bar function for a deliberate subset of the lookback bars.

14.20.518. 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 no 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.

14.20.519. 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 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 * **FillPrice**.

A more complex example for less liquid futures, would be: *Max(2 * TickSize, 0.02 * 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: *Max(0.05, 0.002 * FillPrice, 0.1 * Shares / 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".

14.20.520. 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 0 to *count*-1 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**.

Example

```
Trend: Slope(C, 20)
```

Positive slope indicates an upward trend over 20 bars.

14.20.521. 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 approach to correlation.

This implementation handles ties correctly. Output will match Python **scipy.spearmanr**.

A typical use case is **this indicator**, i.e., *100 * Spearman(C, FunBar, 10).*

See also **Correl**, **CorrelAvg**, etc.

Example

```
RankCorr: Spearman(C, V, 20)
```

Rank correlation between price and volume over 20 bars.

14.20.522. 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**.

Example

```
SplitDay: Split <> 1
```

Non-1 value on a bar indicates a stock split. Use with caution to avoid look-ahead bias.

14.20.523. Sqr

Category

General-Purpose Functions

Description

Square Root of a number

Syntax

Sqr(value)

Parameters

value - formula specifying a numeric value

Example

```
Volatility: Sqr(252) * StdDev(Log(C / C[1]), 20)
```

Annualized volatility from daily log returns.

14.20.524. StartDate (Import)

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.)

The default interpretation of *Earliest* can be changed if desired by editing RealTest.ini (when RT is not running) and changing *EarliestImportYear*.

See also **EndDate**and **NumBars**.

14.20.525. StartDate (Setting)

Category

Settings

Description

The first date to include in a scan or test

Choices

Date Constant - a literal date

Earliest - the oldest date of any symbol in the current data file

Notes

If a date range is not specified in a script then the dates from the **Settings Panel** will be used.

If **DateSym** is specified then *Earliest* means the oldest date of that specific symbol.

14.20.526. StartPercent

Category

Strategy Elements

Description

Percent of initial **AccountSize** to give to this strategy

Input

Any formula specifying a percentage (evaluated once at the start of the test)

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.

Specifying *StartPercent* below 100 also changes the strategy's default **Allocation** from *Combined(S.Equity)* to the strategy's own *S.Equity*, so that the strategy compounds on its own sub-account. Percent-based position sizing and capacity constraints such as **MaxExposure** are then measured against the strategy's own allocation rather than total account value. For example, *MaxExposure: 100* means 100% of the strategy's slice, not 100% of the portfolio.

Note that *StartPercent: 100* is equivalent to not specifying *StartPercent* at all; separate-account modeling requires a value below 100.

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.

14.20.527. 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

The strategies in a *StatsGroup* are specified with the **Using** statement.

StatsGroups can include **Strategies** or **Benchmarks** but cannot include other *StatsGroups*.

A *Strategy* or *Benchmark* can be included in any number of *StatsGroups*.

See **Special Strategy Types** for details.

14.20.528. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

See also **Skewness** and **Kurtosis**.

Example

```
SD20: StdDev(ROC(C, 1), 20)
```

Standard deviation of daily returns over 20 bars.

14.20.529. 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 0 to *count*-1 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, divided by (count - 2).

This function is equivalent to =STEYX in Excel.

Example

```
Parameters:
    len:      20
    mult:     2

Data:
    lreg:     LinReg(C, len)
    serr:     StdErr(C, len)
    upper:    lreg + mult * serr
    lower:    lreg - mult * serr

Charts:
    lreg:     lreg
    upper:    upper
    lower:    lower
```

Example

```
SE20: StdErr(C, 20)
```

Measures how closely price follows a linear trend over 20 bars.

14.20.530. 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)).*

Example

```
SlowK: STOC(14, 3)
```

14.20.531. 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 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.

14.20.532. StratData

Category

Script Sections

Description

Named formulas calculated and stored in memory arrays for each strategy each date while tests are run

Notes

See **TestData and StratData Sections** for a more detailed description.

14.20.533. Strategy

Category

Script Sections

Description

Defines a trading strategy

Notes

See **Strategy Section** for details.

14.20.534. StrategyScore

Category

Strategy Elements

Description

Provides a value to use for this strategy when ranking all strategies to determine setup prioritization

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.

This contest is among the strategies that share an entry phase. To give one strategy absolute priority over another - its orders fully finalized before the other's setups are even evaluated - use **EntryPhase** instead.

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**

14.20.535. StratNum

Category

Current Strategy Information

Description

Ordinal number of the current strategy in the script

Notes

This can be useful in a **StrategyScore** formula.

See also **?Strategy**, which returns the current strategy name, and **StratRef**, which allows dynamic strategy lookup.

14.20.536. StratRef

Category

General-Purpose Functions

Description

Find the number of a given strategy, and/or allow dynamic external symbol usage

Syntax

StratRef(strategy)

Parameters

strategy - any formula returning a string or a number

Return Value

The **StratNum** of the specified strategy, or an empty (nan) result if the strategy is not found

Notes

If the current strategy is "mr_long", then *@mr_long*, *StratNum*, *StratRef(StratNum)*, *StratRef(?Strategy)*, and *StratRef("mr_long")* will each return the same number.

StratRef can also be used as the first argument to **Extern**, for cases where you need to dynamically refer to other strategies.

Example

```
OtherEquity: Extern(StratRef("LongOnly"), S.Equity)
```

StratRef returns the strategy number, enabling dynamic Extern references.

14.20.537. 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 **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
Vol20: Sum(V, 20)
```

Total volume over the past 20 bars.

14.20.538. 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 **one-pass calculation** when used in the **Data Section** without the optional arguments.

Example

```
GapGain: SumSince(O > C[1] * 1.02, ROC(C, 1))
```

Sum of daily returns since the most recent 2%+ gap up.

14.20.539. 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.

Example

```
SumSqRet: SumSQ(ROC(C, 1), 20)
```

14.20.540. SuperTrend

Category

Indicator Functions

Description

Olivier Seban's SuperTrend indicator

Syntax

SuperTrend(len, mult)

Parameters

len - ATR lookback length

mult - ATR multiplier

Notes

SuperTrend is like a trailing stop above or below typical price.

Typical price (TP) is (H + L) / 2.

The trend line is TP +/- ATR(len) * mult.

When the close crosses above trend, trend flips to TP - ATR * mult.

When the close crosses below trend, trend flips to TP + ATR * mult.

Trend direction is therefore defined by whether the close is above or below trend.

ATR is calculated using Wilder's smoothing.

Examples

```
ST: SuperTrend(10, 3)
ExitStop: ST
```

SuperTrend works well as a trailing stop level.

14.20.541. Switch

Category

General-Purpose Functions

Description

Value switch function

Syntax

Switch(value, match1, value1, match2, value2, ...)

Parameters

value - the value to switch on

match1...matchN - values to compare it to

value1...valueN - value to return for corresponding match

Notes

Switch(Symbol, $AAPL, 1, $GOOGL, 2, $MSFT, 3)

is equivalent to:

Select(Symbol = $AAPL, 1, Symbol = $GOOGL, 2, Symbol = $MSFT, 3)

or to:

if(Symbol = $AAPL, 1, if(Symbol = $GOOGL, 2, if(Symbol = $MSFT, 3, nan)))

As with **Select**, the final *match, value* pair can be replaced by a single *value* to specify what to return if there are no matches, e.g.

Switch(Symbol, $AAPL, 1, $GOOGL, 2, 3)

would return 3 for all other symbols, whereas the prior example would return *nan* if there were no matches.

Arguments can be numeric or strings. If strings then comparison is case-insensitive.

Example

```
Weight: Switch(DayOfWeek, 1, 0.8, 5, 0.8, 1.0)
```

Returns 0.8 for Monday (1) or Friday (5), 1.0 as default for other days.

14.20.542. 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 constant **Symbol References** like **$MSFT** to compare specific symbols to the value of this variable.

See also **?Symbol**, which returns the current symbol as a string, and **SymRef**, 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.

14.20.543. 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 on or before 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_ndu.csv* file that is included with RealTest in the Examples folder:

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: Examples\ibfutures_ndu.csv* to your settings.

If using CSI futures data, use *Examples\ibfutures_csi.csv* instead.

For an overview of how renamed, delisted and temporarily recoded symbols are handled in live trading (including the automatic asset ID matching that makes this list unnecessary for most Norgate symbol changes), see **Symbol Changes in Open Positions**.

14.20.544. 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 Files** for details.

14.20.545. SymRef

Category

General-Purpose Functions

Description

Find the number of a given symbol, and/or allow dynamic external symbol usage

Syntax

SymRef(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*, *SymRef(Symbol)*, *SymRef(?Symbol)*, and *SymRef("MSFT")* will each return the same number.

SymRef can also be used as the first argument to **Extern**, for cases where you need to dynamically refer to other symbols.

Example

```
OtherClose: Extern(SymRef("AAPL"), C)
```

SymRef returns the symbol number, enabling dynamic Extern references.

14.20.546. T.Bars

Category

Trade Record Values

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.

14.20.547. T.CommIn

Category

Trade Record Values

Description

Commission paid for the entry transaction of a trade, in dollars

14.20.548. T.CommOut

Category

Trade Record Values

Description

Commission paid for the exit transaction of a trade, in dollars

14.20.549. T.DateIn

Category

Trade Record Values

Description

Entry date of a trade as a number in yyyymmdd format

14.20.550. T.DateOut

Category

Trade Record Values

Description

Exit date of a trade as a number in yyyymmdd format

14.20.551. T.Div

Category

Trade Record Values

Description

Net dividend received or paid over the duration of a trade ($; sign follows trade side, and includes cash-in-lieu of fractional split shares)

14.20.552. T.Fraction

Category

Trade Record Values

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.

14.20.553. T.FxIn

Category

Trade Record Values

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.

14.20.554. T.FxOut

Category

Trade Record Values

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.

14.20.555. T.Highest

Category

Trade Record Values

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.

14.20.556. T.Lowest

Category

Trade Record Values

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.

14.20.557. T.NetFx

Category

Trade Record Values

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.

14.20.558. T.NetPct

Category

Trade Record Values

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 / (T.QtyIn * T.PriceIn * T.FxIn) / T.PtVal*

See also **T.Profit**

14.20.559. T.Points

Category

Trade Record Values

Description

The net points (split-adjusted $/share) gained or lost in a trade (negative for a loss)

14.20.560. T.PriceIn

Category

Trade Record Values

Description

Trade entry price

14.20.561. T.PriceInZ

Category

Trade Record Values

Description

Theoretical zero-slippage entry price

Notes

See **TLCalcZPrices** for a description of this feature.

14.20.562. T.PriceOut

Category

Trade Record Values

Description

Trade exit price

14.20.563. T.PriceOutZ

Category

Trade Record Values

Description

Theoretical zero-slippage exit price

Notes

See **TLCalcZPrices** for a description of this feature.

14.20.564. T.Profit

Category

Trade Record Values

Description

The net trade profit or loss of a trade after commission and dividend are applied, expressed in dollars

Notes

See also **T.NetPct**

14.20.565. T.PtVal

Category

Trade Record Values

Description

The **point value** of symbol of a trade

14.20.566. T.QtyIn

Category

Trade Record Values

Description

The number of shares or contracts bought or shorted when a trade was entered

14.20.567. T.QtyOut

Category

Trade Record Values

Description

The number of shares or contracts sold or covered when a trade was exited

14.20.568. T.Reason

Category

Trade Record Values

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
7	reduce position
8	zero position
9	roll forward

Setup/Entry Skip Reason Codes:

Code	Reason
1	max invested
2	max positions
3	max entries
4	max same category
5	max same symbol
6	zero quantity
7	limit not hit
8	stop not hit
9	ambiguous timing
10	entry skip
11	max setups
12	max exposure
13	max new positions
14	max new exposure
15	max new invested
16	stop+limit not hit
17	no data
18	zero volume
19	setup skip
20	max long exposure
21	max long invested
22	max net exposure
23	max net invested
24	max short exposure
25	max short invested
26	min free cash
27	min net exposure
28	min net invested
29	max symbol exposure

30	max symbol invested
31	max category exposure
32	max category invested
33	end of test
34	open position (not a skip: still open at end of test)
35	missing bar (AllowMissingBar: False)
36	bad price (fill price would be NaN or zero)

14.20.569. T.RollCosts

Category

Trade Record Values

Description

Total roll costs paid for a futures trade, in dollars.

Notes

Use the **RollCost** strategy formula to specify the cost of each roll transaction (both sides).

Use **S.RollCosts** to see the total roll costs for all trades closed that day.

14.20.570. T.Side

Category

Trade Record Values

Description

The side of a trade (1=long, -1=short)

14.20.571. T.SlipIn

Category

Trade Record Values

Description

The amount of slippage applied to the entry price of a trade, in total dollars

14.20.572. T.SlipOut

Category

Trade Record Values

Description

The amount of slippage applied to the exit price of a trade, in total dollars

14.20.573. T.SplitIn

Category

Trade Record Values

Description

The bar split factor (actual price / adjusted price) at trade entry time

14.20.574. T.SplitOut

Category

Trade Record Values

Description

The bar split factor (actual price / adjusted price) at trade exit time

14.20.575. T.Strat

Category

Trade Record Values

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

14.20.576. T.Symbol

Category

Trade Record Values

Description

The symbol number of a trade

Notes

Symbols are numbered internally by the order in which they were loaded. T.Symbol also carries the trade's ticker symbol, so in a string context it yields the symbol name (like ?Symbol for the current bar) and can be used in comparisons and output.

14.20.577. T.TimeIn

Category

Trade Record Values

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.

14.20.578. T.TimeOut

Category

Trade Record Values

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.

14.20.579. T.ValueIn

Category

Trade Record Values

Description

Value calculated by **EntryTradeValue** when position was entered

Notes

This item can be useful in **Trade Statistics Functions**

14.20.580. T.ValueOut

Category

Trade Record Values

Description

Value calculated by **ExitTradeValue** when position was exited

Notes

This item can be useful in **Trade Statistics Functions**

14.20.581. 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).

Example

```
TanVal: Tangent(45)
```

Returns 1. Parameter is in degrees.

14.20.582. TanH

Category

General-Purpose Functions

Description

Hyperbolic Tangent of a number

Syntax

TanH(value)

Parameters

value - formula

Notes

The TanH function normalizes financial time series to a range of -1 to 1, aiding in creating smooth, bounded trading indicators and improving the detection of overbought and oversold conditions. Its properties include being linear in the middle, which helps in interpreting small changes clearly, and bounding outliers, which prevents extreme values from disproportionately affecting the analysis.

Example

```
Normalized: TanH((C - MA(C, 20)) / StdDev(C, 20))
```

Normalizes a z-score to the -1 to 1 range, compressing outliers.

14.20.583. 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.

Example

```
EntryLimit: TargetPrice(RSI(2), 10)
```

The price that, if it were tomorrow's close, would cause RSI(2) to equal 10.

14.20.584. TEMA

Category

Multi-Bar Functions

Description

Triple Exponential Moving Average

Syntax

TEMA(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

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 / (count + 1).

Triple EMA is calculated as *3 * EMA(expr, count) - 3 * EMA(EMA(expr, count), count) + EMA(EMA(EMA(expr, count), count), count)*

Unlike the **EMA** function, *TEMA* applies the exponential weight factor from the start of available data rather than beginning with *MA(expr, count)*. This is consistent with how other backtesting software calculates this indicator.

This function supports **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
TripleMA: TEMA(C, 21)
```

14.20.585. Template

Category

Script Sections

Description

Defines a strategy template

Notes

See **Special Strategy Types** for details.

14.20.586. TestInclude

Category

Script Sections

Description

Allows a script to include another script when run in **Test Mode** or **Optimize 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.

14.20.587. TestData

Category

Script Sections

Description

Named formulas calculated and stored in memory arrays for each symbol each date while tests are run

Notes

See **TestData and StratData Sections** for a more detailed description.

14.20.588. 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**.

14.20.589. 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

Trades - open the **trade list**

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 - show output from **DebugEntry**, **DebugExit** and **DebugTargetStop** statements

Notes

If *TestOutput* is not specified in a script then the choices from the **Settings Panel** will be used.

14.20.590. 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.

14.20.591. 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.

14.20.592. TestScanPositions

Category

Settings

Description

Runs the TestScan for all open positions rather than for all symbols

Choices

True - scan open positions

False - (default) scan all symbols

Notes

See **Test Output Scan** for more information.

14.20.593. 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.

14.20.594. 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 what 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.

This can also be used to anchor an entire sub-expression within any multi-bar function, as in this example:

```
MinCorr: Lowest(This(Correl(C, C[FunBar], 20)), 10)
```

Here we are doing what would typically require a loop construct in a one-line function call.

Anchoring to the current bar, this calculates the 20-bar auto-correlation of Close with lags of 1 to 10 bars and returns the lowest correlation value found.

FunBar is deliberately ignored by the Correl function itself since it is often used as one of the arguments to represent a linear series. It is therefore applied to the outer function (Lowest) instead.

Without the *This* anchor, the function would correlate C to C[1], then C[1] to C[3], then C[2] to C[5], and so on.

14.20.595. 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 indicator or other data-value calculations - formulas always see the full precision of the imported bar data. It is, however, the default value of **PriceRound**, so by default trade prices are rounded to the as-traded *TickSize* before profits and losses are computed.

14.20.596. 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.

14.20.597. TLAdoptPos

Category

Strategy Elements

Description

Specifies whether trades from a **TradeList** that were entered prior to the test's **StartDate** should be "adopted" by the strategy

Choices

False - do not adopt positions entered before the start date (default)

True - do adopt them

Notes

Trades entered before the start date will only be adopted by the strategy if they were exited after the start date or have no exit specified.

When such trades are adopted, their entry dates are changed to match the test's start date.

14.20.598. TLCalcZPrices

Category

Strategy Elements

Description

Specifies whether theoretical zero-slippage entry and exit prices are calculated when **Hybrid Trade List** strategies are run in Orders mode.

Choices

False - do not calculate zero-slippage prices (default)

True - do calculate them

Notes

When a strategy that is traded with **OrderClerk** or that includes an explicit **TradeList** is run in Orders mode, the trade list is "played back" verbatim to accurately establish current positions before generating tomorrow's orders using the strategy's formulas.

To facilitate the calculation of live-trade slippage by comparing each actual fill price to the theoretical zero-slippage price, RealTest can optionally evaluate the relevant strategy formulas for every past trade and store the theoretical prices in the **T.PriceInZ** and **T.PriceOutZ** fields of the trade records.

For example, for a position entered with a limit order, the theoretical fill price would be the limit price that was used, unless the open already touched that price in which case it would be the open.

When TLCalcZPrices is True, RealTest will also calculate **T.SlipIn** and **T.SlipOut** for each trade by comparing T.PriceInZ to **T.PriceIn** and T.PriceOutZ to **T.PriceOut**.

Sorting the test's trade list by SlipIn or SlipOut makes it easy to see the slippage outliers at either end of the column and the median values near its middle.

TradeStatAvg and **TradeStatStdDev** can be used in **Results** formulas to summarize these stats for all the trades in a live-trade playback test.

14.20.599. 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 *TLDateFmt* is to use the *Date Input* 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.

14.20.600. TLDelim

Category

Strategy Elements

Description

Specifies the column delimiter in a **TradeList** file.

Choices

Comma

Semicolon

Tab

Notes

The default column delimiter is *Comma* when **TLNumFmt** is *Point* or *Semicolon* when *TLNumFmt* is *Comma*.

14.20.601. TLEndDate

Category

Strategy Elements

Description

The date to switch from pure **TradeList** playback to pure strategy formula processing.

Choices

0 - (default) play back the trade list and superimpose strategy formulas (if provided) on those imported trades

Latest - play back the trade list verbatim and ignore the strategy definition

any date - play back the trade list verbatim up through that date, then ignore the rest of the imported trades and apply the strategy definition thereafter

Notes

The default "0" choice (or omitting this element) runs a *TradeList* strategy as was always done prior to release 2.0.29.2. The **EntrySetup** formula is ignored. Other strategy formulas, if specified, will modify the behavior of the imported trade.

The "Latest" choice, or specifying any date after the backtest end date, causes a pure trade list playback regardless of what else is in the strategy definition.

Specifying a date that is earlier than the backtest end date runs a pure trade list playback through that date and then switches to a pure strategy backtest.

This strategy setting only applies when **run mode** is *Test.*

In *Orders* run mode, the trade list (if present) is played back verbatim until the test end date and then the strategy rules are applied to generate tomorrow's orders in so-called **Hybrid Trade List** mode.

Another way to think of *TLEndDate* is that it enables *hybrid trade list* playback in *Test* mode.

14.20.602. TLFields

Category

Strategy Elements

Description

Imported trade list CSV field order specification (comma-separated list)

Choices

AssetId - the stock's asset ID from Norgate or any other source with persistent asset IDs (takes precedence over Symbol, so trades are matched correctly even if the symbol has since changed; if the asset ID matches no loaded stock, the Symbol column is used instead)

OrderClerk 1.0.29.5 and later supply this column automatically in OrderClerkTrades.csv; see

Symbol Changes in Open Positions for the complete round trip.

Symbol - the stock symbol

Underlying - the underlying symbol; treated as an alias for Symbol (useful for option or IB flex-query trade lists)

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

Reason - a string of up to 80 characters describing the exit reason of the trade

TradeID - an optional numeric trade identifier stored with the trade (round-trip trade lists only)

Notes

The *TLFields* order does not need to be specified if the **TradeList** file's first row is a header using recognized field names (such as Symbol, DateIn, PriceIn); in that case RealTest auto-detects the column layout from that header. Specify TLFields explicitly when the file has no header row or uses non-standard column names.

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.

14.20.603. TLNumFmt

Category

Strategy Elements

Description

Specifies whether numbers (prices) in a **TradeList** use a point or a comma as their decimal separator

Choices

Point - point (period) is used (default)

Comma - comma is used

Notes

By default comma is the field separator in a CSV trade list file, and point is the decimal separator.

This optional item enables European-format "CSV" trade lists to be imported by RealTest.

Specify *TLNumFmt: Comma* if your trade lists use comma in place of point as the decimal delimiter.

When using *TLNumFmt: Comma*, be sure to also specify SemiColon as your **TLDelim**.

14.20.604. 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.

14.20.605. 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.

14.20.606. 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.

14.20.607. 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.

14.20.608. 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 (the / may instead be - or .)
- 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.

Example

```
RefDate: ToDate("01/15/2024")
```

Parses a date string and returns a numeric date value (20240115).

14.20.609. ToLower

Category

String Functions

Description

Convert a string to all lowercase

Syntax

ToLower(string)

Parameters

string - a **literal string** or **string function** result

Example

```
LowerSym: ToLower(?Symbol)
```

14.20.610. 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.

Example

```
FirstNum: ToNum(?Name, 1)
```

Finds the first numeric value within a string.

14.20.611. ToUpper

Category

String Functions

Description

Convert a string to all uppercase

Syntax

ToUpper(string)

Parameters

string - a **literal string** or **string function** result

Example

```
UpperName: ToUpper(?Name)
```

14.20.612. 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).*

Example

```
EconSect: Top(InfoTRBC, 2)
```

Extract the 2-digit economic sector from a 10-digit TRBC code.

14.20.613. Tracker

Category

Strategy Elements

Description

Specifies whether to automatically create a tracking benchmark copy of this strategy.

Choices

None - do not add a tracking benchmark (default)

Hidden - add a tracking benchmark that will not appear in graphs or reports

Visible - add a tracking benchmark that will appear in graphs and reports

Notes

When *Tracker* is specified (other than *None*), a **Benchmark** copy of the current strategy is internally created.

Specifying *Tracker: Hidden* names the auto-generated benchmark as the strategy name with a leading underscore and with "_track" appended, e.g. the tracker for a strategy called "mr_long" would be added as "_mr_long_track".

Specifying *Tracker: Visible* does the same, just without the leading underscore. (A leading underscore on a benchmark name means "do not show this benchmark in graphs or reports".)

Use **IsTracker** in your strategy+tracker formulas to know which one is being evaluated.

See the **mr_sample_tracking.rts** example script for a simple example of *Tracker* and *IsTracker* usage.

See the **oex_tf_auto_track.rts** example script and its description in the **StratData** section topic for a more complex example.

14.20.614. 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, unless the CSV file's header row already uses the standard field names.

See **Using an Imported Trade List** for a detailed description of this mechanism.

14.20.615. Trades

Category

Script Sections

Description

Backtest trade list column definitions

Notes

See **Trades Section** and **Trade List Windows**.

14.20.616. 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 or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

See the link above for important information about this function category.

Examples

```
AvgProfit: TradeStatAvg(T.Profit)
```

Average profit of all closed trades.

```
RecentAvg: TradeStatAvg(T.NetPct, 10)
```

Average return of the 10 most recent trades.

14.20.617. TradeStatKurt

Category

Trade Statistics Functions

Description

The kurtosis of trade record values for the most recent N trades or for all trades

Syntax

TradeStatKurt(value, count, symbol)

Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

This is excess kurtosis, so a normal distribution returns 0. Positive values indicate fatter tails (more frequent extreme trades) than a normal distribution.

TradeStatKurt is calculated in the way that Excel would calculate a KURT.P function if it had one, i.e., as if the set of trade values is the entire population. It returns excess kurtosis (a normal distribution is 0).

See the link above for important information about this function category.

Example

```
ReturnKurt: TradeStatKurt(T.NetPct)
```

Excess kurtosis of the distribution of trade returns.

14.20.618. 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 or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

See the link above for important information about this function category.

Examples

```
LongestTrade: TradeStatMax(T.Bars)
```

Duration of the longest closed trade.

```
BestReturn: TradeStatMax(T.NetPct, 20)
```

Best return among the 20 most recent trades.

14.20.619. 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 or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

See the link above for important information about this function category.

Example

```
WorstReturn: TradeStatMin(T.NetPct)
```

Worst return among all closed trades.

14.20.620. TradeStatSkew

Category

Trade Statistics Functions

Description

The skewness of trade record values for the most recent N trades or for all trades

Syntax

TradeStatSkew(value, count, symbol)

Parameters

value - trade value formula

count - number of recent trades to include, or all trades if omitted or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

Skewness measures the asymmetry of the trade value distribution: positive values indicate a longer right tail (a few unusually large winners), negative values a longer left tail (a few unusually large losers).

TradeStatSkew is calculated in the same way that Excel calculates the SKEW.P function, which is as if the set of trade values is the entire population.

See the link above for important information about this function category.

Example

```
ReturnSkew: TradeStatSkew(T.NetPct)
```

Skewness of the distribution of trade returns.

14.20.621. 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 or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

TradeStatStdDev is calculated in the same way that Excel calculates the STDEV.P function, which is as if the set of trade values is the entire population.

See the link above for important information about this function category.

Example

```
TradeVol: TradeStatStdDev(T.NetPct)
```

Volatility (standard deviation) of trade returns.

14.20.622. 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 or 0

symbol - specific symbol to include, or all symbols if omitted

Notes

See the link above for important information about this function category.

Examples

```
TotalProfit: TradeStatSum(T.Profit)
```

Total dollar profit of all closed trades.

```
SymProfit: TradeStatSum(T.Profit, 0, Symbol)
```

Total profit for the current symbol only. Count=0 means all trades.

14.20.623. 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.

Example

```
LastTrough: Trough(L, 10)
```

Most recent low followed by a 10% rise. Does not look ahead.

14.20.624. 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.

Example

```
BarsSinceTrough: TroughBars(L, 10)
```

14.20.625. TrueInRow

Category

Multi-Bar Functions

Description

Count of bars in a row for which a condition was true

Syntax

TrueInRow(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 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 **one-pass calculation** when used in the **Data Section** without a count argument.

Example

```
ConsecUp: TrueInRow(C > C[1])
```

Count of consecutive up days ending today (0 if today is a down day).

14.20.626. 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**.

14.20.627. UntilEvent

Category

General-Purpose Functions

Description

Number of market days from the current bar until the next instance of an event of the given type

Syntax

UntilEvent(type)

Parameters

type - event type number or quoted event type name (same types used with **Event**)

Notes

Returns 1 if the next event of this type is on tomorrow's bar, 2 if it is the market day after, and so on. Only events dated strictly after the current bar are counted, so an event falling on the current bar is not seen; if no later event of this type exists for the current symbol (including when a quoted name does not appear in the event list), the function returns -1.

Conceptually equivalent to *UntilTrue(Event(type))*, with one critical difference: *UntilEvent* reads the underlying event list directly, so it sees events dated after the last imported bar.

This makes the function suitable for live order generation: it can report, for example, "7 market days until the next earnings release" when the release date is published in the **Event List File** but lies past the most recent price bar. **UntilTrue** cannot do this because it only looks at loaded bar data.

For events that have already occurred relative to the current bar, use *Event(type)* directly, or

Event(type, 1) to get the most recent value.

Example

```
DaysToEarnings: UntilEvent(1)
SkipNearEarnings: DaysToEarnings >= 0 and DaysToEarnings <= 3
```

Avoid taking new positions within three market days of a scheduled earnings release.

14.20.628. 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 - look-ahead 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 **one-pass calculation** when used in the **Data Section** without a count argument.

Example

```
DaysToFriday: UntilTrue(DayOfWeek = 5)
```

WARNING: this function looks into the future. Generally for date-related usage, not price data.

14.20.629. 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.

14.20.630. 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 however 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**.

14.20.631. 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 or 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.*

Using has a second role: when it appears in a **StatsGroup** section it does not copy elements, but instead lists the strategies and/or benchmarks that make up the group (a StatsGroup may not include a Template or another StatsGroup).

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.

14.20.632. Volume or V

Category

Bar Data Values

Description

Current bar volume

Notes

Either *Volume* or *V* can be used as the name of this value.

14.20.633. VWMA

Category

Multi-Bar Functions

Description

Volume-Weighted Moving Average

Syntax

VWMA(expr, count)

Parameters

expr - data series formula

count - lookback period

Notes

VWMA is calculated as *Sum(expr * Volume, count) / Sum(Volume, count)*

VWMA(Close, count) is a way to calculate volume-weighted average price (VWAP).

This function supports **one-pass calculation** when used in the **Data Section** with a non-variable count.

Example

```
VWAP20: VWMA(C, 20)
```

VWMA(C, count) calculates a volume-weighted average price.

14.20.634. 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 **Multiple Tests and Optimization** 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.

A hand-written *WalkForward* section containing only a *Dates* line with two dates runs an explicit in-sample / out-of-sample split test; see **In-Sample / Out-Of-Sample**.

See *spy_tlt_uis.rts* in the *Examples* folder for an example that uses walk-forward optimization to implement a strategy from an article.

14.20.635. Week

Category

Bar Data Values

Description

Current bar's week of the year (1-54)

Notes

This function works identically to =*WEEKNUM(cell, 2)* in Excel.

Neither Excel nor RealTest follow the ISO-8601 standard for week-of-year numbering.

14.20.636. 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 **one-pass calculation** when used in the **Data Section** without the optional arguments.

Example

```
GapOpen: WhenTrue(O > C[1] * 1.02, O)
```

The opening price on the most recent 2%+ gap-up day.

14.20.637. 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.

Example

```
WeightedMA: WMA(C, 20)
```

WMA and WAvg are interchangeable names.

14.20.638. 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.

14.20.639. 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 0 to *count*-1 is used if omitted, so YInt is the value at the oldest bar in the window)

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**.

Example

```
YI20: YInt(C, 20)
```

Y-intercept of 20-bar linear regression of closing prices.

Index

#

#Avg, 313
#ByCII, 314
#ByEcon, 314
#ByGroup, 314
#ByIndu, 315
#ByListNum, 315
#ByMkt, 316
#BySect, 316
#Count, 316
#DataValueFile, 317
#DenseRank, 317
#DVFAlign, 318
#DVFFill, 318
#DVFPad, 321
#Fill, 318
#Highest, 319
#Lowest, 319
#Median, 319
#OnePerDate, 320
#OnePerSym, 320
#Pad, 321
#PercentRank, 321
#PercentRankN, 322
#Rank, 322
#SlowCalc, 323
#StdDev, 323
#Sum, 323

?

?AccountSize, 324
?CII, 324
?Currency, 324
?Domicile, 324
?EconSect, 325
?EndDate, 325
?EquityType, 325
?Exchange, 325
?IndGroup, 326
?Industry, 326
?ListingType, 326
?LocalTime, 326
?Name, 327
?OrdersDate, 327
?ReportingCurrency, 327
?RunMode, 328
?ScriptName, 328
?Sector, 328
?StartDate, 329
?Strategy, 328

?StratType, 329
?Symbol, 329
?Type, 330

A

Abs, 330
AccountSize, 330
AccountType, 331
Adjusted, 332
Adjustment, 331
ADX, 333
AEMA, 333
AESD, 334
Allocation, 334
AllowMissingBar, 335
AllowNoVolume, 336
AllowSameName, 336
AlwaysImport, 337
Ambiguity, 337
ArcCos, 337
ArcSin, 338
ArcTan, 338
Assert, 339
ATR, 340
Avg, 446

B

BarDate, 341
BarNum, 341
BarsHeld, 341
BarSize, 342
BarsLeft, 342
BarStart, 343
BBBOT, 343
BBBotF, 343
BBPCT, 344
BBPctF, 345
BBTOP, 345
BBTopF, 346
BBTREND, 346
BBTrendF, 347
BBWIDTH, 347
BBWidthF, 348
Benchmark, 348
Bound, 349

C

CalendarSym, 349
CashInOut, 350
CashIntPct, 350
CashList, 351
Category, 352
CCI, 352
CDF, 353

Charts, 354
CIIFamily, 354
CIILevel, 355
Classification, 355
Close, 356
CloseSlip, 356
Combined, 357, 358
CommCurrency, 358
Commission, 358
Compounded, 359
Constituency, 359
Contracts, 546
Correl, 361
CorrelAvg, 361
CorrelMax, 363
CorrelMaxSym, 364
CorrelMed, 362
CorrelMin, 362
CorrelMinSym, 363
Cosine, 364
CountTrue, 365
Cross, 365
CRSI, 366
CSVDateFmt, 366
CSVDelim, 367
CSVFields, 367
CSVFile, 369
CSVNumFmt, 369
Currency, 369

D

Data, 370
DataFile, 370
DataPath, 370
DataSource, 371
DataType, 371
DataValueFile, 372
Date, 372
DateBars, 373
DateDay, 373
DateDisplay, 373
DateInput, 374
DateMonth, 374
DateSym, 374
DateYear, 375
Day, 375
DayOfWeek, 375
DayOfYear, 376
Days, 376
DaysPerYear, 377
DebugEntry, 378
DebugEval, 378
DebugExit, 379

DebugTargetStop, 379
DEMA, 380
Dividend, 380
DividendsToCash, 377
DllDataCalc, 381
DynamicSizing, 381

E

EhlersBP, 382
EhlersHP, 383
EhlersSS, 384
EhlersUS, 384
EMA, 385
EndDate, 386
EndOfMonth, 386
EndOfTestExits, 387
EndOfWeek, 388
EntryDate, 388
EntryLimit, 389
EntryPhase, 389
EntryRank, 390
EntryScore, 391
EntrySetup, 391
EntrySkip, 392
EntryStop, 392
EntryTime, 393
EntryTradeValue, 394
ESD, 395
EventListFile, 396
ExchangeMap, 396
ExcludeIf, 399
ExcludeList, 399
ExitLimit, 400
ExitLimitQty, 400
ExitLimitTime, 401
ExitNum, 402
ExitQty, 402
ExitRank, 403
ExitRule, 403
ExitScore, 404
ExitStop, 404
ExitStopQty, 405
ExitStopTime, 405
ExitTime, 406
ExitTradeValue, 407
ExitType, 407
Exp, 407
Extern, 408
Extra, 409

F

F.xxx, 409
FeesInOut, 410
FeesList, 411

FillFraction, 411
FillPrice, 412
FillPriceAvg, 412
FillPriceMax, 412
FillPriceMin, 413
FillQty, 413
FillValue, 414
FilterNum, 414
Find, 414
Format, 415
FormatDate, 415
FunBar, 417
Fundamentals, 416
FunStat, 418
FuturesMargin, 356
FxRatio, 418

G

GetField, 419
Graphs, 420
GreedyScoring, 419

H

HAvg, 421
HHV, 420
High, 420
Highest, 420
HMA, 421
HolidayList, 421
Hurst, 422
HVOL, 423

I

ICDF, 423
IF, 424
IgnoreDividends, 424
IIF, 424
Import, 425
Include, 425
IncludeList, 426
InfoDelist, 427
InfoExpiry, 427
InfoFloat, 428
InfoGICS, 428
InfoID, 427
InfoMargin, 429
InfoShares, 429
InfoTRBC, 429
InList, 430
InSet, 430
InXXX, 427
IsExit, 431
IsNaN, 431
IsOrder, 433

IsSetup, 433
IsTracker, 434
Item, 434

J

JDSAR, 435

K

KAMA, 436
KBBOT, 436
KBTOP, 437
KeepAdjusted, 437
KeepNonIncluded, 438
KeepRedundant, 438
KeepTrades, 439
Kurtosis, 439

L

Left, 440
Length, 440
Library, 441
LimitExtra, 441
LimitSlip, 442
LinReg, 442
ListNum, 443
LLV, 446
Log, 443
LogDetail, 443
LogFile, 444
Logit, 445
Low, 446
Lowest, 446

M

MA, 446
MACD, 447
MACDH, 447
MACDS, 448
MarginIntPct, 448
MarkToMarket, 449
Match, 449
Max, 450
MaxCatExp, 451
MaxCatInv, 452
MaxEntries, 452
MaxExposure, 453
MaxInvested, 453
MaxLongExp, 454
MaxLongInv, 454
MaxN, 451
MaxNetExp, 455
MaxNetInv, 456
MaxNewExp, 456
MaxNewInv, 456

MaxNewPos, 457
MaxPerTurn, 457
MaxPositions, 458
MaxSameCat, 458
MaxSameSym, 459
MaxSetups, 459
MaxShortExp, 459
MaxShortInv, 460
MaxSymExp, 461
MaxSymInv, 461
MDI, 462
Median, 462
Mid, 463
Min, 463
MinFreeCash, 464
MinN, 463
MinNetExp, 465
MinNetInv, 465
MissingBars, 466
Month, 466
MostRecent, 467

N

Namespace, 468
NextOpen, 469
NoNan, 469
Notes, 470
NoWeekends, 470
NthOfMonth, 470
NumBars, 471

O

OBV, 472
Open, 472
OpenSlip, 472
OptAnchor, 473
OptByStrategy, 473
OptBySymbol, 473
OptClearResults, 474
OptContinueResults, 474
OptimizeMode, 474
OptimizeSettings, 475
OptIterations, 475
OptKeepBest, 476
OptKeepByDate, 476
OptKeepBySymbol, 476
OptKeepNoTrades, 476
OptMultiSaveAs, 477
OptMutatePct, 477
OptNoDialog, 477
OptNoStats, 477
OptOverfit, 478
OptOverfitBlocks, 478
OptScoreCol, 478

OptShards, 479
OptShowResults, 479
OptSortResults, 479
OptSummaryLog, 480
OptTestInterval, 480
OptTestLength, 480
OptTimeUnit, 480
OptWalkForward, 481
OrderClerkFolder, 481
OrderExtra, 481
OrderNote, 481
OrderPrice, 483
OrderRank, 483
OrdersClassDelim, 484
OrdersComment, 484
OrderSettings, 485
OrdersFile, 485
OrdersFreeCash, 485
OrdersIgnoreSplits, 486
OrdersInclude, 486
OrdersLiveData, 487
OrdersMktAsLmtPct, 487
OrdersMode, 488
OrdersNetLiq, 488
OrdersTemplate, 489
OrderSum, 491
OrdersUseQtyIn, 490
OrdersUseTLEndDate, 490

P

PadAlignSym, 491
Padding, 492
Parameters, 492
PctChg, 511
PDI, 493
Peak, 493
PeakBars, 494
PercentRank, 495
PercentRankN, 495
PointValue, 496
PositionSum, 498
PrevExitLimit, 496
PrevExitStop, 497
PriceRound, 498
Product, 497

Q

QtyFinal, 499
QtyPrice, 500
QtyRound, 501
QtyType, 501
Quantity, 502

R

Random, 502
RandomSeed, 503
Range, 504
Rank, 504
RankN, 504
Reason, 505
Reduce, 505
RemoveLatest, 506
RepairData, 506
Replace, 507
ReportContent, 508
ReportsFolder, 508
Rescale, 508
Results, 509
ResultsByYear, 509
ResultsFile, 510
Right, 510
RiskFreeRateSym, 511
ROC, 511
RollCost, 511
Round, 512
RRSI, 513
RSI, 513
RsiF, 514

S

S.Alloc, 514
S.BPM, 515
S.BPQ, 515
S.BPW, 515
S.BPY, 515
S.Cash, 520
S.CashInOut, 516
S.Comms, 516
S.Compounded, 516
S.Date, 517
S.DDAmt, 517
S.DDBars, 517
S.DDDlr, 517
S.DDPct, 517
S.Divs, 518
S.Entries, 518
S.EntryOrders, 518
S.Equity, 519
S.Exits, 519
S.Exposure, 519
S.FeesInOut, 520
S.First, 520
S.FreeCash, 520
S.FuturesMargin, 516
S.Interest, 521
S.Invested, 521
S.LongExp, 521

S.LongInv, 522
S.LongM2M, 522
S.LossAmt, 522
S.LossBars, 523
S.LossDlr, 522
S.Losses, 523
S.LossPct, 523
S.LossPctAlloc, 523
S.M2M, 523
S.MAE, 524
S.MarkedToMarket, 524
S.MaxAlloc, 525
S.MaxDDAmt, 525
S.MaxDDBars, 525
S.MaxDDDlr, 525
S.MaxDDPct, 525
S.MaxEquity, 526
S.MFE, 524
S.MinAlloc, 526
S.MinEquity, 526
S.NetAmt, 526
S.NetDlr, 526
S.NetFx, 527
S.NetPct, 528
S.Number, 529
S.Positions, 529
S.RiskFreeRate, 529
S.RollCosts, 530
S.Setups, 530
S.ShortExp, 530
S.ShortInv, 530
S.ShortM2M, 531
S.Slips, 531
S.StartEquity, 531
S.Stops, 532
S.Targets, 532
S.TradeAmt, 532
S.TradeBars, 532
S.TradeDlr, 532
S.TradePct, 532
S.TradePctAlloc, 533
S.TWEQ, 533
S.Usage, 533
S.WinAmt, 534
S.WinBars, 534
S.WinDlr, 534
S.WinPct, 534
S.WinPctAlloc, 534
S.Wins, 534
SAR, 535
SarF, 535
SaveAs, 536
SaveChartsTo, 537

SaveLogAs, 536
SavePositionsAs, 537
SaveScanAs, 537
SaveStatsAs, 539
SaveStatsInclude, 539
SaveTestListAs, 539
SaveTradesAs, 540
SaveTradesType, 540
Scan, 541
ScanInclude, 541
ScanNoDefCols, 541
ScanNoHeader, 541
ScanNoWindow, 542
ScanSettings, 542
Select, 542
Sequence, 543
Settings, 544
SetupRank, 544
SetupScore, 544
SetupSkip, 545
SetupSum, 545
Shares, 546
ShortLoanIntPct, 546
ShortPrcdsIntPct, 546
Side, 547,
Sign, 547
SigSpike, 548
SinceHigh, 548
SinceLow, 549
SinceTrue, 549
Sine, 550
Skewness, 550
SkipNans, 551
SkipTestIf, 551
Slippage, 552
Slope, 553
Spearman, 553
Split, 554
Sqr, 554
StartDate, 555
StartPercent, 555
StatsGroup, 556
StdDev, 556
StdErr, 557
STOC, 558
StopSlip, 559
StratData, 559
Strategy, 559
StrategyScore, 559
StratNum, 560
StratRef, 561
Sum, 561
SumSQ, 562

SuperTrend, 563
Switch, 563
Symbol, 564
SymChangeList, 564
SymInfoFile, 566
SymNum, 566
SymRef, 566

T

T.Bars, 566
T.CommIn, 567
T.CommOut, 567
T.DateIn, 567
T.DateOut, 567
T.Div, 567
T.Fraction, 568
T.FxIn, 568
T.FxOut, 568
T.Highest, 568
T.Lowest, 569
T.NetFx, 569
T.NetPct, 569
T.Points, 569
T.PriceIn, 570
T.PriceInZ, 570
T.PriceOut, 570
T.PriceOutZ, 570
T.Profit, 571
T.PtVal, 571
T.QtyIn, 571
T.QtyOut, 571
T.Reason, 571
T.RollCosts, 573
T.Side, 573
T.SlipIn, 573
T.SlipOut, 573
T.SplitIn, 574
T.SplitOut, 574
T.Strat, 574
T.Symbol, 574
T.TimeIn, 574
T.TimeOut, 575
T.ValueIn, 575
T.ValueOut, 575
Tangent, 575
TanH, 576
TargetPrice, 576
TEMA, 577
Template, 578
TestData, 578
TestInclude, 578
TestName, 579
TestOutput, 579

TestScan, 579
TestScanAllDates, 580
TestScanPositions, 580
TestSettings, 580
Text, 415
This, 580
TickSize, 581
TLAdjusted, 582
TLAdoptPos, 582
TLCalcZPrices, 582
TLDateFmt, 583
TLDelim, 583
TLEndDate, 584
TLFields, 584
TLNumFmt, 586
TLStratName, 586
TLTimeShift, 586
TLValueIn, 587
TLValueOut, 587
ToDate, 587
ToLower, 588
ToNum, 588
Top, 589
ToUpper, 589
TR, 596
Tracker, 590
TradeList, 590
Trades, 591
TradeStatAvg, 591
TradeStatKurt, 591
TradeStatMax, 592
TradeStatMin, 592
TradeStatSkew, 593
TradeStatStdDev, 593
TradeStatSum, 594
Trough, 594
TroughBars, 595
TrueInRow, 595
TrueRange, 596

U

UntilEvent, 596
UntilTrue, 597
Update, 597
UseAvailableBars, 598
Using, 598

V

Volume, 599
VWMA, 599

W

WalkForward, 600
WAvg, 601

Week, 600
WhenTrue, 600
WMA, 601

X

XAvg, 385

Y

Year, 601
YInt, 602