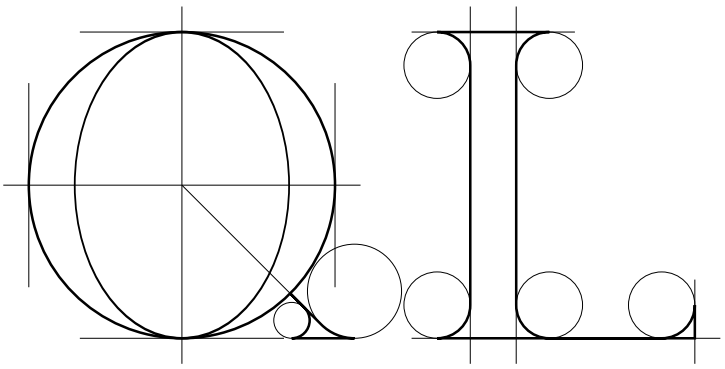


QuantLib

An open source library for quantitative finance

Version 0.3.4

Generated by Doxygen 1.3.4

18 Nov 2003

Contents

I User Manual 1

1 An introduction to QuantLib 3

1.1 Introduction . . . 3
1.2 Project overview . . . 4
1.3 Where to get QuantLib . . . 6
1.4 Installation . . . 7
1.5 Usage . . . 8
1.6 Supported platforms . . . 9
1.7 Version history . . . 10
1.8 Todo List . . . 20
1.9 Additional resources . . . 22
1.10 The QuantLib Group . . . 23
1.11 Copyright and License . . . 24

2 QuantLib components 27

2.1 Core classes . . . 27
2.2 Date and time calculations . . . 28
2.3 Lattice methods . . . 29
2.4 The finite differences framework . . . 31
2.5 The Monte Carlo framework . . . 36
2.6 The short-rate modelling framework . . . 42
2.7 Currencies and FX rates . . . 44
2.8 Instruments and pricers . . . 45
2.9 Math tools . . . 47
2.10 Design patterns . . . 49
2.11 Term structures . . . 50
2.12 Utilities . . . 51

3 Examples 53

3.1 QuantLib Examples 53

II Reference Manual 55

4 QuantLib Module Index 57
4.1 QuantLib Modules 57

5 QuantLib Namespace Index 59
5.1 QuantLib Namespace List 59

6 QuantLib Hierarchical Index 61
6.1 QuantLib Class Hierarchy 61

7 QuantLib Class Index 73
7.1 QuantLib Class List 73

8 QuantLib File Index 83
8.1 QuantLib File List 83

9 QuantLib Module Documentation 95
9.1 Global QuantLib macros 95
9.2 Math functions 97
9.3 Numeric limits 98
9.4 Time functions 99
9.5 String functions 100
9.6 Character functions 101
9.7 Input/output functions 102
9.8 Min and max functions 103
9.9 Template capabilities 104
9.10 Iterator support 106

10 QuantLib Namespace Documentation 107
10.1 QuantLib Namespace Reference 107
10.2 QuantLib::Calendars Namespace Reference 117
10.3 QuantLib::CashFlows Namespace Reference 119
10.4 QuantLib::DayCounters Namespace Reference 124
10.5 QuantLib::FiniteDifferences Namespace Reference 125
10.6 QuantLib::Indexes Namespace Reference 127
10.7 QuantLib::Instruments Namespace Reference 128

10.8 QuantLib::Math Namespace Reference . . . 131
10.9 QuantLib::MonteCarlo Namespace Reference . . . 135
10.10 QuantLib::Patterns Namespace Reference . . . 138
10.11 QuantLib::Pricers Namespace Reference . . . 139
10.12 QuantLib::RandomNumbers Namespace Reference . . . 142
10.13 QuantLib::ShortRateModels Namespace Reference . . . 144
10.14 QuantLib::Solvers1D Namespace Reference . . . 147
10.15 QuantLib::TermStructures Namespace Reference . . . 148
10.16 QuantLib::Utilities Namespace Reference . . . 150
10.17 QuantLib::VolTermStructures Namespace Reference . . . 151

11 QuantLib Class Documentation **153**
11.1 Actual360 Class Reference . . . 153
11.2 Actual365 Class Reference . . . 154
11.3 ActualActual Class Reference . . . 155
11.4 AcyclicVisitor Class Reference . . . 156
11.5 AdditiveEQPBinomialTree Class Reference . . . 157
11.6 AffineModel Class Reference . . . 158
11.7 AffineTermStructure Class Reference . . . 159
11.8 AmericanCondition Class Reference . . . 161
11.9 AmericanExercise Class Reference . . . 162
11.10 AmericanMCVanillaEngine Class Reference . . . 163
11.11 AnalyticalCapFloor Class Reference . . . 164
11.12 AnalyticAmericanBinaryEngine Class Reference . . . 165
11.13 AnalyticBarrierEngine Class Reference . . . 166
11.14 AnalyticEuropeanBinaryEngine Class Reference . . . 167
11.15 AnalyticEuropeanEngine Class Reference . . . 168
11.16 Arguments Class Reference . . . 169
11.17 ArithmeticAPOPathPricer_old Class Reference . . . 170
11.18 ArithmeticASOPathPricer_old Class Reference . . . 171
11.19 ArmijoLineSearch Class Reference . . . 172
11.20 Array Class Reference . . . 173
11.21 ArrayFormatter Class Reference . . . 176
11.22 AssertionFailedError Class Reference . . . 177
11.23 AssetOrNothingPayoff Class Reference . . . 178
11.24 AUDLibor Class Reference . . . 179
11.25 Barrier Struct Reference . . . 180

11.26 BarrierEngine Class Reference . . . 181
11.27 BarrierOption Class Reference . . . 182
11.28 BarrierOption Class Reference . . . 184
11.29 BarrierOption::arguments Class Reference . . . 185
11.30 BarrierOption::results Class Reference . . . 186
11.31 BarrierPathPricer Class Reference . . . 187
11.32 BasketPathPricer_old Class Reference . . . 188
11.33 BermudanExercise Class Reference . . . 189
11.34 BiasedBarrierPathPricer Class Reference . . . 190
11.35 BicubicSplineInterpolation Class Template Reference . . . 191
11.36 BilinearInterpolation Class Template Reference . . . 193
11.37 Binary Struct Reference . . . 195
11.38 BinaryEngine Class Reference . . . 196
11.39 BinaryOption Class Reference . . . 197
11.40 BinaryOption Class Reference . . . 199
11.41 BinaryOption::arguments Class Reference . . . 200
11.42 BinaryOption::results Class Reference . . . 201
11.43 BinaryPathPricer Class Reference . . . 202
11.44 BinomialTree Class Reference . . . 203
11.45 BinomialVanillaEngine Class Reference . . . 204
11.46 Bisection Class Reference . . . 205
11.47 BlackCapFloor Class Reference . . . 206
11.48 BlackConstantVol Class Reference . . . 207
11.49 BlackKarasinski Class Reference . . . 209
11.50 BlackKarasinski::Dynamics Class Reference . . . 210
11.51 BlackModel Class Reference . . . 211
11.52 BlackScholesLattice Class Reference . . . 213
11.53 BlackScholesProcess Class Reference . . . 214
11.54 BlackSwaption Class Reference . . . 215
11.55 BlackVarianceCurve Class Reference . . . 216
11.56 BlackVarianceSurface Class Reference . . . 218
11.57 BlackVarianceTermStructure Class Reference . . . 220
11.58 BlackVolatilityTermStructure Class Reference . . . 221
11.59 BlackVolTermStructure Class Reference . . . 222
11.60 BoundaryCondition Class Template Reference . . . 224
11.61 BoundaryConstraint Class Reference . . . 226

11.62 BoxMullerGaussianRng Class Template Reference . . . 227
11.63 BPSCalculator Class Reference . . . 228
11.64 Brent Class Reference . . . 229
11.65 Bridge Class Template Reference . . . 230
11.66 BrownianBridge Class Template Reference . . . 231
11.67 BSMOperator Class Reference . . . 232
11.68 Budapest Class Reference . . . 233
11.69 CADLibor Class Reference . . . 234
11.70 Calendar Class Reference . . . 235
11.71 Calendar::WesternImpl Class Reference . . . 238
11.72 CalendarImpl Class Reference . . . 239
11.73 CalibrationHelper Class Reference . . . 240
11.74 CalibrationSet Class Reference . . . 242
11.75 Cap Class Reference . . . 243
11.76 CapFlatVolatilityStructure Class Reference . . . 244
11.77 CapFlatVolatilityVector Class Reference . . . 246
11.78 CapFloor Class Reference . . . 247
11.79 CapFloor::arguments Class Reference . . . 249
11.80 CapFloor::results Class Reference . . . 250
11.81 CapletForwardVolatilityStructure Class Reference . . . 251
11.82 CashFlow Class Reference . . . 252
11.83 CashOrNothingPayoff Class Reference . . . 254
11.84 CHFLibor Class Reference . . . 255
11.85 CLGaussianRng Class Template Reference . . . 256
11.86 CliquetEngine Class Reference . . . 257
11.87 CliquetOption Class Reference . . . 258
11.88 CliquetOption Class Reference . . . 259
11.89 CliquetOption::arguments Class Reference . . . 260
11.90 CliquetOptionPathPricer Class Reference . . . 261
11.91 CliquetOptionPathPricer_old Class Reference . . . 262
11.92 Collar Class Reference . . . 263
11.93 combining_iterator Class Template Reference . . . 264
11.94 CompositeConstraint Class Reference . . . 266
11.95 CompositeMarketElement Class Template Reference . . . 267
11.96 ConjugateGradient Class Reference . . . 268
11.97 ConstantParameter Class Reference . . . 269

11.98 Constraint Class Reference . . . 270
11.99 ConstraintImpl Class Reference . . . 271
11.100 ContinuousGeometricAPO Class Reference . . . 272
11.101 CostFunction Class Reference . . . 273
11.102 coupling_iterator Class Template Reference . . . 274
11.103 Coupon Class Reference . . . 276
11.104 CoxIngersollRoss Class Reference . . . 278
11.105 CoxIngersollRoss::Dynamics Class Reference . . . 280
11.106 CoxRossRubinstein Class Reference . . . 281
11.107 CrankNicolson Class Template Reference . . . 282
11.108 CubicSpline Class Reference . . . 283
11.109 CubicSplineInterpolation Class Template Reference . . . 284
11.110 CumulativeNormalDistribution Class Reference . . . 285
11.111 CuriouslyRecurringTemplate Class Template Reference . . . 286
11.112 CurrencyFormatter Class Reference . . . 287
11.113 Date Class Reference . . . 288
11.114 DateFormatter Class Reference . . . 291
11.115 DayCounter Class Reference . . . 292
11.116 DayCounterImpl Class Reference . . . 294
11.117 DepositRateHelper Class Reference . . . 295
11.118 DerivedMarketElement Class Template Reference . . . 297
11.119 DiffusionProcess Class Reference . . . 298
11.120 DirichletBC Class Reference . . . 300
11.121 DiscountCurve Class Reference . . . 301
11.122 DiscountStructure Class Reference . . . 303
11.123 DiscrepancyStatistics Class Reference . . . 305
11.124 DiscreteGeometricAPO Class Reference . . . 306
11.125 DiscreteGeometricASO Class Reference . . . 307
11.126 DiscretizedAsset Class Reference . . . 308
11.127 DiscretizedDiscountBond Class Reference . . . 310
11.128 DiscretizedOption Class Reference . . . 311
11.129 Disposable Class Template Reference . . . 313
11.130 DMinus Class Reference . . . 314
11.131 DoubleFormatter Class Reference . . . 315
11.132 DPlus Class Reference . . . 316
11.133 DPlusDMinus Class Reference . . . 317

11.134 DriftTermStructure Class Reference . . . 318
11.135 DZero Class Reference . . . 320
11.136 EndCriteria Class Reference . . . 321
11.137 EqualJumpsBinomialTree Class Reference . . . 323
11.138 EqualProbabilitiesBinomialTree Class Reference . . . 324
11.139 Error Class Reference . . . 325
11.140 ErrorFunction Class Reference . . . 326
11.141 Euribor Class Reference . . . 327
11.142 EuroFormatter Class Reference . . . 328
11.143 EuropeanExercise Class Reference . . . 329
11.144 EuropeanOption Class Reference . . . 330
11.145 EuropeanPathPricer Class Reference . . . 331
11.146 EuropeanPathPricer_old Class Reference . . . 332
11.147 EverestPathPricer_old Class Reference . . . 333
11.148 Exercise Class Reference . . . 334
11.149 ExplicitEuler Class Template Reference . . . 335
11.150 ExtendedCoxIngersollRoss Class Reference . . . 336
11.151 ExtendedCoxIngersollRoss::Dynamics Class Reference . . . 338
11.152 ExtendedCoxIngersollRoss::FittingParameter Class Reference . . . 339
11.153 ExtendedDiscountCurve Class Reference . . . 340
11.154 FalsePosition Class Reference . . . 342
11.155 FdAmericanOption Class Reference . . . 343
11.156 FdBermudanOption Class Reference . . . 344
11.157 FdBsmOption Class Reference . . . 345
11.158 FdDividendEuropeanOption Class Reference . . . 347
11.159 FdDividendShoutOption Class Reference . . . 348
11.160 FdEuropean Class Reference . . . 349
11.161 FdStepConditionOption Class Reference . . . 350
11.162 filtering_iterator Class Template Reference . . . 351
11.163 FiniteDifferenceModel Class Template Reference . . . 352
11.164 FixedRateCoupon Class Reference . . . 353
11.165 FloatingRateCoupon Class Reference . . . 355
11.166 Floor Class Reference . . . 357
11.167 ForwardEngine Class Template Reference . . . 358
11.168 ForwardOptionArguments Class Template Reference . . . 359
11.169 ForwardPerformanceEngine Class Template Reference . . . 360

11.170 ForwardRateStructure Class Reference . . . 361
11.171 ForwardSpreadedTermStructure Class Reference . . . 363
11.172 ForwardVanillaOption Class Reference . . . 365
11.173 Frankfurt Class Reference . . . 367
11.174 FraRateHelper Class Reference . . . 368
11.175 FuturesRateHelper Class Reference . . . 370
11.176 G2 Class Reference . . . 371
11.177 G2::FittingParameter Class Reference . . . 373
11.178 GammaFunction Class Reference . . . 374
11.179 GaussianStatistics Class Template Reference . . . 375
11.180 GBPLibor Class Reference . . . 377
11.181 GeneralStatistics Class Reference . . . 378
11.182 GenericEngine Class Template Reference . . . 381
11.183 GenericModelEngine Class Template Reference . . . 382
11.184 GenericRiskStatistics Class Template Reference . . . 383
11.185 GeometricAPOPathPricer_old Class Reference . . . 386
11.186 GeometricASOPathPricer_old Class Reference . . . 387
11.187 Greeks Class Reference . . . 388
11.188 HaltonRsg Class Reference . . . 389
11.189 Handle Class Template Reference . . . 390
11.190 Helsinki Class Reference . . . 392
11.191 HimalayaPathPricer_old Class Reference . . . 393
11.192 History Class Reference . . . 394
11.193 History::const_iterator Class Reference . . . 397
11.194 History::Entry Class Reference . . . 399
11.195 HullWhite Class Reference . . . 400
11.196 HullWhite::Dynamics Class Reference . . . 402
11.197 HullWhite::FittingParameter Class Reference . . . 403
11.198 ICGaussianRng Class Template Reference . . . 404
11.199 ICGaussianRsg Class Template Reference . . . 405
11.200 IllegalArgumentError Class Reference . . . 406
11.201 IllegalResultError Class Reference . . . 407
11.202 ImplicitEuler Class Template Reference . . . 408
11.203 ImpliedTermStructure Class Reference . . . 409
11.204 ImpliedVolTermStructure Class Reference . . . 411
11.205 InArrearIndexedCoupon Class Reference . . . 413

11.206 IncrementalStatistics Class Reference 414
11.207 Index Class Reference 417
11.208 IndexedCoupon Class Reference 419
11.209 IndexError Class Reference 421
11.210 Instrument Class Reference 422
11.211 IntegerFormatter Class Reference 425
11.212 IntegralEngine Class Reference 426
11.213 Interpolation Class Template Reference 427
11.214 Interpolation2D Class Template Reference 429
11.215 InverseCumulativeNormal Class Reference 431
11.216 JamshidianSwaption Class Reference 432
11.217 JarrowRudd Class Reference 433
11.218 Johannesburg Class Reference 434
11.219 JointCalendar Class Reference 435
11.220 JPYLibor Class Reference 436
11.221 KnuthUniformRng Class Reference 437
11.222 KronrodIntegral Class Reference 438
11.223 Lattice Class Reference 439
11.224 Lattice2D Class Reference 441
11.225 LatticeShortRateModelEngine Class Template Reference 442
11.226 LazyObject Class Reference 443
11.227 LeastSquareFunction Class Reference 446
11.228 LeastSquareProblem Class Reference 447
11.229 LecuyerUniformRng Class Reference 448
11.230 LexicographicalView Class Template Reference 449
11.231 Linear Class Reference 451
11.232 LinearInterpolation Class Template Reference 452
11.233 LineSearch Class Reference 453
11.234 Link Class Template Reference 455
11.235 LocalConstantVol Class Reference 457
11.236 LocalVolCurve Class Reference 459
11.237 LocalVolSurface Class Reference 461
11.238 LocalVolTermStructure Class Reference 463
11.239 LogLinear Class Reference 465
11.240 LogLinearInterpolation Class Template Reference 466
11.241 London Class Reference 467

11.242 lowest_category_iterator Struct Template Reference 468
11.243 MakeSchedule Class Reference 469
11.244 MarketElement Class Reference 470
11.245 Matrix Class Reference 471
11.246 MaxBasketPathPricer_old Class Reference 474
11.247 MCBarrierEngine Class Template Reference 475
11.248 McBasket Class Reference 477
11.249 MCBinaryEngine Class Template Reference 478
11.250 McCliquetOption Class Reference 480
11.251 McDiscreteArithmeticAPO Class Reference 481
11.252 McDiscreteArithmeticASO Class Reference 482
11.253 McEuropean Class Reference 483
11.254 MCEuropeanEngine Class Template Reference 484
11.255 McEverest Class Reference 485
11.256 McHimalaya Class Reference 486
11.257 McMaxBasket Class Reference 487
11.258 McPagoda Class Reference 488
11.259 McPerformanceOption Class Reference 489
11.260 McPricer Class Template Reference 490
11.261 McSimulation Class Template Reference 492
11.262 MCVanillaEngine Class Template Reference 494
11.263 MersenneTwisterUniformRng Class Reference 496
11.264 Method Class Reference 497
11.265 Milan Class Reference 499
11.266 MixedScheme Class Template Reference 500
11.267 Model Class Reference 502
11.268 MonteCarloModel Class Template Reference 504
11.269 MoroInverseCumulativeNormal Class Reference 505
11.270 MultiPath Class Reference 506
11.271 MultiPathGenerator Class Template Reference 507
11.272 MultiPathGenerator_old Class Template Reference 508
11.273 MultivariateAccumulator Class Reference 509
11.274 NeumannBC Class Reference 511
11.275 Newton Class Reference 512
11.276 NewtonSafe Class Reference 513
11.277 NewYork Class Reference 514

11.278 NoConstraint Class Reference . . . 515
11.279 NonLinearLeastSquare Class Reference . . . 516
11.280 NormalDistribution Class Reference . . . 518
11.281 Null Class Template Reference . . . 519
11.282 NullCalendar Class Reference . . . 520
11.283 NullParameter Class Reference . . . 521
11.284 NumericalMethod Class Reference . . . 522
11.285 Observable Class Reference . . . 523
11.286 Observer Class Reference . . . 525
11.287 OneFactorAffineModel Class Reference . . . 528
11.288 OneFactorModel Class Reference . . . 529
11.289 OneFactorModel::ShortRateDynamics Class Reference . . . 530
11.290 OneFactorModel::ShortRateTree Class Reference . . . 531
11.291 OneFactorOperator Class Reference . . . 532
11.292 Option Class Reference . . . 533
11.293 OrnsteinUhlenbeckProcess Class Reference . . . 534
11.294 Oslo Class Reference . . . 536
11.295 OutOfMemoryError Class Reference . . . 537
11.296 PagodaPathPricer_old Class Reference . . . 538
11.297 Parameter Class Reference . . . 539
11.298 ParameterImpl Class Reference . . . 540
11.299 ParCoupon Class Reference . . . 541
11.300 Path Class Reference . . . 543
11.301 PathGenerator Class Template Reference . . . 544
11.302 PathGenerator_old Class Template Reference . . . 545
11.303 PathPricer Class Template Reference . . . 546
11.304 PathPricer_old Class Template Reference . . . 547
11.305 Payoff Class Reference . . . 548
11.306 PerformanceOption Class Reference . . . 549
11.307 PerformanceOptionPathPricer_old Class Reference . . . 550
11.308 Period Class Reference . . . 551
11.309 PiecewiseConstantParameter Class Reference . . . 552
11.310 PiecewiseFlatForward Class Reference . . . 553
11.311 PlainVanillaPayoff Class Reference . . . 555
11.312 PositiveConstraint Class Reference . . . 556
11.313 PostconditionNotSatisfiedError Class Reference . . . 557

11.314 PreconditionNotSatisfiedError Class Reference . . . 558
11.315 PricingEngine Class Reference . . . 559
11.316 PrimeNumbers Class Reference . . . 560
11.317 Problem Class Reference . . . 561
11.318 processing_iterator Class Template Reference . . . 563
11.319 QuantoEngine Class Template Reference . . . 565
11.320 QuantoForwardVanillaOption Class Reference . . . 566
11.321 QuantoOptionArguments Class Template Reference . . . 568
11.322 QuantoOptionResults Class Template Reference . . . 569
11.323 QuantoTermStructure Class Reference . . . 570
11.324 QuantoVanillaOption Class Reference . . . 572
11.325 RandomArrayGenerator Class Template Reference . . . 574
11.326 RandomSequenceGenerator Class Template Reference . . . 575
11.327 RateFormatter Class Reference . . . 576
11.328 RateHelper Class Reference . . . 577
11.329 RelinkableHandle Class Template Reference . . . 579
11.330 Results Class Reference . . . 581
11.331 Ridder Class Reference . . . 582
11.332 RiskMeasures Class Reference . . . 583
11.333 RiskStatistics_old Class Reference . . . 585
11.334 Sample Struct Template Reference . . . 587
11.335 Schedule Class Reference . . . 588
11.336 Secant Class Reference . . . 589
11.337 SegmentIntegral Class Reference . . . 590
11.338 SequenceStatistics Class Template Reference . . . 591
11.339 Short Class Template Reference . . . 593
11.340 ShortFloatingRateCoupon Class Reference . . . 594
11.341 ShoutCondition Class Reference . . . 595
11.342 SimpleCashFlow Class Reference . . . 596
11.343 SimpleDayCounter Class Reference . . . 597
11.344 SimpleMarketElement Class Reference . . . 598
11.345 SimpleSwap Class Reference . . . 599
11.346 SimpleSwap::arguments Class Reference . . . 601
11.347 SimpleSwap::results Class Reference . . . 602
11.348 Simplex Class Reference . . . 603
11.349 SimpsonIntegral Class Reference . . . 604

11.350 SingleAssetOption Class Reference . . . 605
11.351 SobolRsg Class Reference . . . 607
11.352 Solver1D Class Template Reference . . . 608
11.353 SquareRootProcess Class Reference . . . 610
11.354 SteepestDescent Class Reference . . . 611
11.355 StepCondition Class Template Reference . . . 612
11.356 stepping_iterator Class Template Reference . . . 613
11.357 Stock Class Reference . . . 615
11.358 Stockholm Class Reference . . . 616
11.359 StrikedTypePayoff Class Reference . . . 617
11.360 StringFormatter Class Reference . . . 618
11.361 SupersharePayoff Class Reference . . . 619
11.362 SVD Class Reference . . . 620
11.363 Swap Class Reference . . . 621
11.364 SwapRateHelper Class Reference . . . 623
11.365 Swaption Class Reference . . . 625
11.366 Swaption::arguments Class Reference . . . 627
11.367 Swaption::results Class Reference . . . 628
11.368 SwaptionVolatilityMatrix Class Reference . . . 629
11.369 SwaptionVolatilityStructure Class Reference . . . 630
11.370 Sydney Class Reference . . . 631
11.371 SymmetricSchurDecomposition Class Reference . . . 632
11.372 TARGET Class Reference . . . 633
11.373 TermStructure Class Reference . . . 634
11.374 TermStructureConsistentModel Class Reference . . . 637
11.375 TermStructureFittingParameter Class Reference . . . 638
11.376 Thirty360 Class Reference . . . 639
11.377 Tian Class Reference . . . 640
11.378 TimeBasket Class Reference . . . 641
11.379 TimeGrid Class Reference . . . 642
11.380 Tokyo Class Reference . . . 643
11.381 Toronto Class Reference . . . 645
11.382 TrapezoidIntegral Class Reference . . . 646
11.383 Tree Class Reference . . . 648
11.384 TreeCapFloor Class Reference . . . 649
11.385 TreeSwaption Class Reference . . . 650

11.386 TridiagonalOperator Class Reference . . . 651
11.387 TridiagonalOperator::TimeSetter Class Reference . . . 653
11.388 Trigeorgis Class Reference . . . 654
11.389 TrinomialBranching Class Reference . . . 655
11.390 TrinomialTree Class Reference . . . 656
11.391 TwoFactorModel Class Reference . . . 657
11.392 TwoFactorModel::ShortRateDynamics Class Reference . . . 658
11.393 TwoFactorModel::ShortRateTree Class Reference . . . 659
11.394 UpFrontIndexedCoupon Class Reference . . . 660
11.395 USDLibor Class Reference . . . 661
11.396 Value Class Reference . . . 662
11.397 VanillaEngine Class Reference . . . 663
11.398 VanillaOption Class Reference . . . 664
11.399 VanillaOption::arguments Class Reference . . . 667
11.400 VanillaOption::results Class Reference . . . 668
11.401 Vasicek Class Reference . . . 669
11.402 Vasicek::Dynamics Class Reference . . . 671
11.403 Visitor Class Template Reference . . . 672
11.404 Warsaw Class Reference . . . 673
11.405 Wellington Class Reference . . . 674
11.406 Xibor Class Reference . . . 675
11.407 XiborManager Class Reference . . . 677
11.408 ZARLibor Class Reference . . . 678
11.409 ZeroCurve Class Reference . . . 679
11.410 ZeroSpreadedTermStructure Class Reference . . . 681
11.411 ZeroYieldStructure Class Reference . . . 683
11.412 Zurich Class Reference . . . 685

12 QuantLib File Documentation **687**

12.1 ql/argsandresults.hpp File Reference . . . 687
12.2 ql/array.hpp File Reference . . . 688
12.3 ql/blackmodel.hpp File Reference . . . 689
12.4 ql/calendar.cpp File Reference . . . 690
12.5 ql/calendar.hpp File Reference . . . 691
12.6 ql/Calendars/budapest.cpp File Reference . . . 692
12.7 ql/Calendars/budapest.hpp File Reference . . . 693
12.8 ql/Calendars/frankfurt.cpp File Reference . . . 694

12.9 ql/Calendars/frankfurt.hpp File Reference . . . 695
12.10 ql/Calendars/helsinki.cpp File Reference . . . 696
12.11 ql/Calendars/helsinki.hpp File Reference . . . 697
12.12 ql/Calendars/johannesburg.cpp File Reference . . . 698
12.13 ql/Calendars/johannesburg.hpp File Reference . . . 699
12.14 ql/Calendars/jointcalendar.cpp File Reference . . . 700
12.15 ql/Calendars/jointcalendar.hpp File Reference . . . 701
12.16 ql/Calendars/london.cpp File Reference . . . 702
12.17 ql/Calendars/london.hpp File Reference . . . 703
12.18 ql/Calendars/milan.cpp File Reference . . . 704
12.19 ql/Calendars/milan.hpp File Reference . . . 705
12.20 ql/Calendars/newyork.cpp File Reference . . . 706
12.21 ql/Calendars/newyork.hpp File Reference . . . 707
12.22 ql/Calendars/nullcalendar.hpp File Reference . . . 708
12.23 ql/Calendars/oslo.cpp File Reference . . . 709
12.24 ql/Calendars/oslo.hpp File Reference . . . 710
12.25 ql/Calendars/stockholm.cpp File Reference . . . 711
12.26 ql/Calendars/stockholm.hpp File Reference . . . 712
12.27 ql/Calendars/sydney.cpp File Reference . . . 713
12.28 ql/Calendars/sydney.hpp File Reference . . . 714
12.29 ql/Calendars/target.cpp File Reference . . . 715
12.30 ql/Calendars/target.hpp File Reference . . . 716
12.31 ql/Calendars/tokyo.cpp File Reference . . . 717
12.32 ql/Calendars/tokyo.hpp File Reference . . . 718
12.33 ql/Calendars/toronto.cpp File Reference . . . 719
12.34 ql/Calendars/toronto.hpp File Reference . . . 720
12.35 ql/Calendars/warsaw.cpp File Reference . . . 721
12.36 ql/Calendars/warsaw.hpp File Reference . . . 722
12.37 ql/Calendars/wellington.cpp File Reference . . . 723
12.38 ql/Calendars/wellington.hpp File Reference . . . 724
12.39 ql/Calendars/zurich.cpp File Reference . . . 725
12.40 ql/Calendars/zurich.hpp File Reference . . . 726
12.41 ql/capvolstructures.hpp File Reference . . . 727
12.42 ql/cashflow.hpp File Reference . . . 728
12.43 ql/CashFlows/basispointsensitivity.cpp File Reference . . . 729
12.44 ql/CashFlows/basispointsensitivity.hpp File Reference . . . 730

12.45 ql/CashFlows/cashflowvectors.cpp File Reference . . . 731
12.46 ql/CashFlows/cashflowvectors.hpp File Reference . . . 732
12.47 ql/CashFlows/coupon.hpp File Reference . . . 733
12.48 ql/CashFlows/fixedratecoupon.hpp File Reference . . . 734
12.49 ql/CashFlows/floatingratecoupon.hpp File Reference . . . 735
12.50 ql/CashFlows/inarrearindexedcoupon.hpp File Reference . . . 736
12.51 ql/CashFlows/indexcashflowvectors.hpp File Reference . . . 737
12.52 ql/CashFlows/indexedcoupon.hpp File Reference . . . 738
12.53 ql/CashFlows/parcoupon.cpp File Reference . . . 739
12.54 ql/CashFlows/parcoupon.hpp File Reference . . . 740
12.55 ql/CashFlows/shortfloatingcoupon.cpp File Reference . . . 741
12.56 ql/CashFlows/shortfloatingcoupon.hpp File Reference . . . 742
12.57 ql/CashFlows/shortindexedcoupon.hpp File Reference . . . 743
12.58 ql/CashFlows/simplecashflow.hpp File Reference . . . 744
12.59 ql/CashFlows/timebasket.cpp File Reference . . . 745
12.60 ql/CashFlows/timebasket.hpp File Reference . . . 746
12.61 ql/CashFlows/upfrontindexedcoupon.hpp File Reference . . . 747
12.62 ql/currency.hpp File Reference . . . 748
12.63 ql/dataformatters.cpp File Reference . . . 749
12.64 ql/dataformatters.hpp File Reference . . . 750
12.65 ql/dataparsers.cpp File Reference . . . 751
12.66 ql/dataparsers.hpp File Reference . . . 752
12.67 ql/date.cpp File Reference . . . 753
12.68 ql/date.hpp File Reference . . . 754
12.69 ql/daycounter.hpp File Reference . . . 755
12.70 ql/DayCounters/actual360.hpp File Reference . . . 756
12.71 ql/DayCounters/actual365.hpp File Reference . . . 757
12.72 ql/DayCounters/actualactual.cpp File Reference . . . 758
12.73 ql/DayCounters/actualactual.hpp File Reference . . . 759
12.74 ql/DayCounters/simpledaycounter.cpp File Reference . . . 760
12.75 ql/DayCounters/simpledaycounter.hpp File Reference . . . 761
12.76 ql/DayCounters/thirty360.cpp File Reference . . . 762
12.77 ql/DayCounters/thirty360.hpp File Reference . . . 763
12.78 ql/diffusionprocess.cpp File Reference . . . 764
12.79 ql/diffusionprocess.hpp File Reference . . . 765
12.80 ql/discretizedasset.cpp File Reference . . . 766

12.81 ql/discretizedasset.hpp File Reference . . . 767
12.82 ql/disposable.hpp File Reference . . . 768
12.83 ql/errors.hpp File Reference . . . 769
12.84 ql/exercise.cpp File Reference . . . 771
12.85 ql/exercise.hpp File Reference . . . 772
12.86 ql/FiniteDifferences/americancondition.hpp File Reference . . . 773
12.87 ql/FiniteDifferences/boundarycondition.cpp File Reference . . . 774
12.88 ql/FiniteDifferences/boundarycondition.hpp File Reference . . . 775
12.89 ql/FiniteDifferences/bsmoperator.cpp File Reference . . . 776
12.90 ql/FiniteDifferences/bsmoperator.hpp File Reference . . . 777
12.91 ql/FiniteDifferences/cranknicolson.hpp File Reference . . . 778
12.92 ql/FiniteDifferences/dminus.hpp File Reference . . . 779
12.93 ql/FiniteDifferences/dplus.hpp File Reference . . . 780
12.94 ql/FiniteDifferences/dplusdminus.hpp File Reference . . . 781
12.95 ql/FiniteDifferences/dzero.hpp File Reference . . . 782
12.96 ql/FiniteDifferences/expliciteuler.hpp File Reference . . . 783
12.97 ql/FiniteDifferences/fdtypedefs.hpp File Reference . . . 784
12.98 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference . . . 785
12.99 ql/FiniteDifferences/impliciteuler.hpp File Reference . . . 786
12.100 ql/FiniteDifferences/mixedscheme.hpp File Reference . . . 787
12.101 ql/FiniteDifferences/onefactoroperator.cpp File Reference . . . 788
12.102 ql/FiniteDifferences/onefactoroperator.hpp File Reference . . . 789
12.103 ql/FiniteDifferences/shoutcondition.hpp File Reference . . . 790
12.104 ql/FiniteDifferences/stepcondition.hpp File Reference . . . 791
12.105 ql/FiniteDifferences/tridiagonaloperator.cpp File Reference . . . 792
12.106 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference . . . 793
12.107 ql/FiniteDifferences/valueatcenter.cpp File Reference . . . 794
12.108 ql/FiniteDifferences/valueatcenter.hpp File Reference . . . 795
12.109 ql/functions/daycounters.cpp File Reference . . . 796
12.110 ql/functions/daycounters.hpp File Reference . . . 797
12.111 ql/functions/mathf.cpp File Reference . . . 798
12.112 ql/functions/mathf.hpp File Reference . . . 799
12.113 ql/functions/vols.cpp File Reference . . . 800
12.114 ql/functions/vols.hpp File Reference . . . 801
12.115 ql/grid.cpp File Reference . . . 802
12.116 ql/grid.hpp File Reference . . . 803

12.117 ql/handle.hpp File Reference . . . 804
12.118 ql/history.hpp File Reference . . . 805
12.119 ql/index.hpp File Reference . . . 806
12.120 ql/Indexes/audlibor.hpp File Reference . . . 807
12.121 ql/Indexes/cadlibor.hpp File Reference . . . 808
12.122 ql/Indexes/chflibor.hpp File Reference . . . 809
12.123 ql/Indexes/euribor.hpp File Reference . . . 810
12.124 ql/Indexes/gbplibor.hpp File Reference . . . 811
12.125 ql/Indexes/jpylibor.hpp File Reference . . . 812
12.126 ql/Indexes/usdlibor.hpp File Reference . . . 813
12.127 ql/Indexes/xibor.cpp File Reference . . . 814
12.128 ql/Indexes/xibor.hpp File Reference . . . 815
12.129 ql/Indexes/xibormanager.cpp File Reference . . . 816
12.130 ql/Indexes/xibormanager.hpp File Reference . . . 817
12.131 ql/Indexes/zarlibor.hpp File Reference . . . 818
12.132 ql/instrument.hpp File Reference . . . 819
12.133 ql/Instruments/barrieroption.cpp File Reference . . . 820
12.134 ql/Instruments/barrieroption.hpp File Reference . . . 821
12.135 ql/Instruments/binaryoption.cpp File Reference . . . 822
12.136 ql/Instruments/binaryoption.hpp File Reference . . . 823
12.137 ql/Instruments/capfloor.cpp File Reference . . . 824
12.138 ql/Instruments/capfloor.hpp File Reference . . . 825
12.139 ql/Instruments/forwardvanillaoption.cpp File Reference . . . 826
12.140 ql/Instruments/forwardvanillaoption.hpp File Reference . . . 827
12.141 ql/Instruments/quantoforwardvanillaoption.cpp File Reference . . . 828
12.142 ql/Instruments/quantoforwardvanillaoption.hpp File Reference . . . 829
12.143 ql/Instruments/quantovanillaoption.cpp File Reference . . . 830
12.144 ql/Instruments/quantovanillaoption.hpp File Reference . . . 831
12.145 ql/Instruments/simpleswap.cpp File Reference . . . 832
12.146 ql/Instruments/simpleswap.hpp File Reference . . . 833
12.147 ql/Instruments/stock.cpp File Reference . . . 834
12.148 ql/Instruments/stock.hpp File Reference . . . 835
12.149 ql/Instruments/swap.cpp File Reference . . . 836
12.150 ql/Instruments/swap.hpp File Reference . . . 837
12.151 ql/Instruments/swaption.cpp File Reference . . . 838
12.152 ql/Instruments/swaption.hpp File Reference . . . 839

12.153 ql/Instruments/vanillaoption.cpp File Reference 840
12.154 ql/Instruments/vanillaoption.hpp File Reference 841
12.155 ql/Lattices/binomialtree.cpp File Reference 842
12.156 ql/Lattices/binomialtree.hpp File Reference 843
12.157 ql/Lattices/bsmlattice.cpp File Reference 844
12.158 ql/Lattices/bsmlattice.hpp File Reference 845
12.159 ql/Lattices/lattice.cpp File Reference 846
12.160 ql/Lattices/lattice.hpp File Reference 847
12.161 ql/Lattices/lattice2d.cpp File Reference 848
12.162 ql/Lattices/lattice2d.hpp File Reference 849
12.163 ql/Lattices/tree.hpp File Reference 850
12.164 ql/Lattices/trinomialtree.cpp File Reference 851
12.165 ql/Lattices/trinomialtree.hpp File Reference 852
12.166 ql/marketelement.hpp File Reference 853
12.167 ql/Math/bicubicsplineinterpolation.hpp File Reference 854
12.168 ql/Math/bilinearinterpolation.hpp File Reference 855
12.169 ql/Math/chisquaredistribution.cpp File Reference 856
12.170 ql/Math/chisquaredistribution.hpp File Reference 857
12.171 ql/Math/cubicspline.hpp File Reference 858
12.172 ql/Math/discrepancystatistics.cpp File Reference 859
12.173 ql/Math/discrepancystatistics.hpp File Reference 860
12.174 ql/Math/errorfunction.cpp File Reference 861
12.175 ql/Math/errorfunction.hpp File Reference 862
12.176 ql/Math/functional.hpp File Reference 863
12.177 ql/Math/gammadistribution.cpp File Reference 864
12.178 ql/Math/gammadistribution.hpp File Reference 865
12.179 ql/Math/gaussianstatistics.hpp File Reference 866
12.180 ql/Math/generalstatistics.cpp File Reference 867
12.181 ql/Math/generalstatistics.hpp File Reference 868
12.182 ql/Math/incrementalstatistics.cpp File Reference 869
12.183 ql/Math/incrementalstatistics.hpp File Reference 870
12.184 ql/Math/interpolation.hpp File Reference 871
12.185 ql/Math/interpolation2D.hpp File Reference 872
12.186 ql/Math/interpolationtraits.hpp File Reference 873
12.187 ql/Math/kronrodintegral.hpp File Reference 874
12.188 ql/Math/lexicographicalview.hpp File Reference 875

12.189 ql/Math/linearinterpolation.hpp File Reference . . . 876
12.190 ql/Math/loglinearinterpolation.hpp File Reference . . . 877
12.191 ql/Math/matrix.cpp File Reference . . . 878
12.192 ql/Math/matrix.hpp File Reference . . . 879
12.193 ql/Math/multivariateaccumulator.cpp File Reference . . . 880
12.194 ql/Math/multivariateaccumulator.hpp File Reference . . . 881
12.195 ql/Math/normaldistribution.cpp File Reference . . . 882
12.196 ql/Math/normaldistribution.hpp File Reference . . . 883
12.197 ql/Math/primenumbers.cpp File Reference . . . 884
12.198 ql/Math/primenumbers.hpp File Reference . . . 885
12.199 ql/Math/riskmeasures.hpp File Reference . . . 886
12.200 ql/Math/segmentintegral.hpp File Reference . . . 887
12.201 ql/Math/sequencestatistics.hpp File Reference . . . 888
12.202 ql/Math/statistics.hpp File Reference . . . 890
12.203 ql/Math/svd.cpp File Reference . . . 891
12.204 ql/Math/svd.hpp File Reference . . . 892
12.205 ql/Math/symmetriceigenvalues.hpp File Reference . . . 893
12.206 ql/Math/symmetricschurdecomposition.cpp File Reference . . . 894
12.207 ql/Math/symmetricschurdecomposition.hpp File Reference . . . 895
12.208 ql/Math/trapezoidintegral.hpp File Reference . . . 896
12.209 ql/MonteCarlo/arithmeticapopathpricer.cpp File Reference . . . 897
12.210 ql/MonteCarlo/arithmeticapopathpricer.hpp File Reference . . . 898
12.211 ql/MonteCarlo/arithmeticasopathpricer.cpp File Reference . . . 899
12.212 ql/MonteCarlo/arithmeticasopathpricer.hpp File Reference . . . 900
12.213 ql/MonteCarlo/barrierpathpricer.cpp File Reference . . . 901
12.214 ql/MonteCarlo/barrierpathpricer.hpp File Reference . . . 902
12.215 ql/MonteCarlo/basketpathpricer.cpp File Reference . . . 903
12.216 ql/MonteCarlo/basketpathpricer.hpp File Reference . . . 904
12.217 ql/MonteCarlo/binarypathpricer.cpp File Reference . . . 905
12.218 ql/MonteCarlo/binarypathpricer.hpp File Reference . . . 906
12.219 ql/MonteCarlo/brownianbridge.hpp File Reference . . . 907
12.220 ql/MonteCarlo/cliquetoptionpathpricer.cpp File Reference . . . 908
12.221 ql/MonteCarlo/cliquetoptionpathpricer.hpp File Reference . . . 909
12.222 ql/MonteCarlo/europeanpathpricer.cpp File Reference . . . 910
12.223 ql/MonteCarlo/europeanpathpricer.hpp File Reference . . . 911
12.224 ql/MonteCarlo/everestpathpricer.cpp File Reference . . . 912

12.225 ql/MonteCarlo/everestpathpricer.hpp File Reference . . . 913
12.226 ql/MonteCarlo/geometricapopathpricer.cpp File Reference . . . 914
12.227 ql/MonteCarlo/geometricapopathpricer.hpp File Reference . . . 915
12.228 ql/MonteCarlo/geometricasopathpricer.cpp File Reference . . . 916
12.229 ql/MonteCarlo/geometricasopathpricer.hpp File Reference . . . 917
12.230 ql/MonteCarlo/getcovariance.hpp File Reference . . . 918
12.231 ql/MonteCarlo/himalayapathpricer.cpp File Reference . . . 919
12.232 ql/MonteCarlo/himalayapathpricer.hpp File Reference . . . 920
12.233 ql/MonteCarlo/maxbasketpathpricer.cpp File Reference . . . 921
12.234 ql/MonteCarlo/maxbasketpathpricer.hpp File Reference . . . 922
12.235 ql/MonteCarlo/mctraits.hpp File Reference . . . 923
12.236 ql/MonteCarlo/mctypedefs.hpp File Reference . . . 924
12.237 ql/MonteCarlo/montecarlomodel.hpp File Reference . . . 925
12.238 ql/MonteCarlo/multipath.hpp File Reference . . . 926
12.239 ql/MonteCarlo/multipathgenerator.hpp File Reference . . . 927
12.240 ql/MonteCarlo/pagodapathpricer.cpp File Reference . . . 928
12.241 ql/MonteCarlo/pagodapathpricer.hpp File Reference . . . 929
12.242 ql/MonteCarlo/path.hpp File Reference . . . 930
12.243 ql/MonteCarlo/pathgenerator.hpp File Reference . . . 931
12.244 ql/MonteCarlo/pathpricer.hpp File Reference . . . 932
12.245 ql/MonteCarlo/performanceoptionpathpricer.cpp File Reference . . . 933
12.246 ql/MonteCarlo/performanceoptionpathpricer.hpp File Reference . . . 934
12.247 ql/MonteCarlo/sample.hpp File Reference . . . 935
12.248 ql/null.hpp File Reference . . . 936
12.249 ql/numericalmethod.hpp File Reference . . . 937
12.250 ql/Optimization/armijo.cpp File Reference . . . 938
12.251 ql/Optimization/armijo.hpp File Reference . . . 939
12.252 ql/Optimization/conjugategradient.cpp File Reference . . . 940
12.253 ql/Optimization/conjugategradient.hpp File Reference . . . 941
12.254 ql/Optimization/constraint.hpp File Reference . . . 942
12.255 ql/Optimization/costfunction.hpp File Reference . . . 943
12.256 ql/Optimization/criteria.hpp File Reference . . . 944
12.257 ql/Optimization/leastsquare.hpp File Reference . . . 945
12.258 ql/Optimization/linesearch.hpp File Reference . . . 946
12.259 ql/Optimization/method.hpp File Reference . . . 947
12.260 ql/Optimization/problem.hpp File Reference . . . 948

12.261 ql/Optimization/simplex.cpp File Reference . . . 949
12.262 ql/Optimization/simplex.hpp File Reference . . . 950
12.263 ql/Optimization/steepestdescent.cpp File Reference . . . 951
12.264 ql/Optimization/steepestdescent.hpp File Reference . . . 952
12.265 ql/option.hpp File Reference . . . 953
12.266 ql/Patterns/bridge.hpp File Reference . . . 954
12.267 ql/Patterns/curiouslyrecurring.hpp File Reference . . . 955
12.268 ql/Patterns/lazyobject.hpp File Reference . . . 956
12.269 ql/Patterns/observable.hpp File Reference . . . 957
12.270 ql/Patterns/visitor.hpp File Reference . . . 958
12.271 ql/payoff.hpp File Reference . . . 959
12.272 ql/Pricers/analyticalcapfloor.cpp File Reference . . . 960
12.273 ql/Pricers/analyticalcapfloor.hpp File Reference . . . 961
12.274 ql/Pricers/barrieroptionpricer.cpp File Reference . . . 962
12.275 ql/Pricers/barrieroptionpricer.hpp File Reference . . . 963
12.276 ql/Pricers/binaryoptionpricer.cpp File Reference . . . 964
12.277 ql/Pricers/binaryoptionpricer.hpp File Reference . . . 965
12.278 ql/Pricers/blackcapfloor.cpp File Reference . . . 966
12.279 ql/Pricers/blackcapfloor.hpp File Reference . . . 967
12.280 ql/Pricers/blackswaption.cpp File Reference . . . 968
12.281 ql/Pricers/blackswaption.hpp File Reference . . . 969
12.282 ql/Pricers/capfloorpricer.cpp File Reference . . . 970
12.283 ql/Pricers/capfloorpricer.hpp File Reference . . . 971
12.284 ql/Pricers/cliquetoption.cpp File Reference . . . 972
12.285 ql/Pricers/continuousgeometricapo.hpp File Reference . . . 973
12.286 ql/Pricers/discretegeometricapo.cpp File Reference . . . 974
12.287 ql/Pricers/discretegeometricapo.hpp File Reference . . . 975
12.288 ql/Pricers/discretegeometricaso.cpp File Reference . . . 976
12.289 ql/Pricers/discretegeometricaso.hpp File Reference . . . 977
12.290 ql/Pricers/europeanoption.cpp File Reference . . . 978
12.291 ql/Pricers/europeanoption.hpp File Reference . . . 979
12.292 ql/Pricers/fdamericanoption.hpp File Reference . . . 980
12.293 ql/Pricers/fdbermudanoption.cpp File Reference . . . 981
12.294 ql/Pricers/fdbermudanoption.hpp File Reference . . . 982
12.295 ql/Pricers/fdbsmoption.cpp File Reference . . . 983
12.296 ql/Pricers/fdbsmoption.hpp File Reference . . . 984

12.297 ql/Pricers/fddividendamericanoption.cpp File Reference . . . 985
12.298 ql/Pricers/fddividendamericanoption.hpp File Reference . . . 986
12.299 ql/Pricers/fddividendeuropeanoption.cpp File Reference . . . 987
12.300 ql/Pricers/fddividendeuropeanoption.hpp File Reference . . . 988
12.301 ql/Pricers/fddividendoption.cpp File Reference . . . 989
12.302 ql/Pricers/fddividendoption.hpp File Reference . . . 990
12.303 ql/Pricers/fddividendshoutoption.cpp File Reference . . . 991
12.304 ql/Pricers/fddividendshoutoption.hpp File Reference . . . 992
12.305 ql/Pricers/fdeuropean.cpp File Reference . . . 993
12.306 ql/Pricers/fdeuropean.hpp File Reference . . . 994
12.307 ql/Pricers/fdmultiperiodoption.cpp File Reference . . . 995
12.308 ql/Pricers/fdmultiperiodoption.hpp File Reference . . . 996
12.309 ql/Pricers/fdshoutoption.hpp File Reference . . . 997
12.310 ql/Pricers/fdstepconditionoption.cpp File Reference . . . 998
12.311 ql/Pricers/fdstepconditionoption.hpp File Reference . . . 999
12.312 ql/Pricers/jamshidianswaption.cpp File Reference . . . 1000
12.313 ql/Pricers/jamshidianswaption.hpp File Reference . . . 1001
12.314 ql/Pricers/mcbasket.cpp File Reference . . . 1002
12.315 ql/Pricers/mcbasket.hpp File Reference . . . 1003
12.316 ql/Pricers/mccliquetoption.cpp File Reference . . . 1004
12.317 ql/Pricers/mccliquetoption.hpp File Reference . . . 1005
12.318 ql/Pricers/mcdiscretearithmeticapo.cpp File Reference . . . 1006
12.319 ql/Pricers/mcdiscretearithmeticapo.hpp File Reference . . . 1007
12.320 ql/Pricers/mcdiscretearithmeticaso.cpp File Reference . . . 1008
12.321 ql/Pricers/mcdiscretearithmeticaso.hpp File Reference . . . 1009
12.322 ql/Pricers/mceuropean.cpp File Reference . . . 1010
12.323 ql/Pricers/mceuropean.hpp File Reference . . . 1011
12.324 ql/Pricers/mceverest.cpp File Reference . . . 1012
12.325 ql/Pricers/mceverest.hpp File Reference . . . 1013
12.326 ql/Pricers/mchimalaya.cpp File Reference . . . 1014
12.327 ql/Pricers/mchimalaya.hpp File Reference . . . 1015
12.328 ql/Pricers/mcmaxbasket.cpp File Reference . . . 1016
12.329 ql/Pricers/mcmaxbasket.hpp File Reference . . . 1017
12.330 ql/Pricers/mcpagoda.cpp File Reference . . . 1018
12.331 ql/Pricers/mcpagoda.hpp File Reference . . . 1019
12.332 ql/Pricers/mcperformanceoption.cpp File Reference . . . 1020

12.333 ql/Pricers/mcperformanceoption.hpp File Reference 1021
12.334 ql/Pricers/mcpricer.hpp File Reference 1022
12.335 ql/Pricers/performanceoption.cpp File Reference 1023
12.336 ql/Pricers/performanceoption.hpp File Reference 1024
12.337 ql/Pricers/singleassetoption.cpp File Reference 1025
12.338 ql/Pricers/singleassetoption.hpp File Reference 1026
12.339 ql/Pricers/swaptionpricer.cpp File Reference 1027
12.340 ql/Pricers/swaptionpricer.hpp File Reference 1028
12.341 ql/Pricers/treecapfloor.cpp File Reference 1029
12.342 ql/Pricers/treecapfloor.hpp File Reference 1030
12.343 ql/Pricers/treeswaption.cpp File Reference 1031
12.344 ql/Pricers/treeswaption.hpp File Reference 1032
12.345 ql/pricingengine.hpp File Reference 1033
12.346 ql/PricingEngines/americanmcengines.cpp File Reference 1034
12.347 ql/PricingEngines/americanmcengines.hpp File Reference 1035
12.348 ql/PricingEngines/analyticamericanbinaryengine.cpp File Reference 1036
12.349 ql/PricingEngines/analyticeuropeanbinaryengine.cpp File Reference 1037
12.350 ql/PricingEngines/analyticeuropeanengine.cpp File Reference 1038
12.351 ql/PricingEngines/barrierengines.hpp File Reference 1039
12.352 ql/PricingEngines/binaryengines.hpp File Reference 1040
12.353 ql/PricingEngines/binomialvanillaengine.cpp File Reference 1041
12.354 ql/PricingEngines/cliquetengines.hpp File Reference 1042
12.355 ql/PricingEngines/discretizedvanillaoption.cpp File Reference 1043
12.356 ql/PricingEngines/discretizedvanillaoption.hpp File Reference 1044
12.357 ql/PricingEngines/forwardengines.hpp File Reference 1045
12.358 ql/PricingEngines/genericengine.hpp File Reference 1046
12.359 ql/PricingEngines/integralengines.cpp File Reference 1047
12.360 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference 1048
12.361 ql/PricingEngines/mcengine.hpp File Reference 1049
12.362 ql/PricingEngines/quantoengines.hpp File Reference 1050
12.363 ql/PricingEngines/vanillaengines.hpp File Reference 1051
12.364 ql/qldefines.hpp File Reference 1052
12.365 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference 1054
12.366 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference 1055
12.367 ql/RandomNumbers/haltonrsg.cpp File Reference 1056
12.368 ql/RandomNumbers/haltonrsg.hpp File Reference 1057

12.369 ql/RandomNumbers/inversecumgaussianrng.hpp File Reference 1058
12.370 ql/RandomNumbers/inversecumgaussianrsg.hpp File Reference 1059
12.371 ql/RandomNumbers/knuthuniformrng.cpp File Reference 1060
12.372 ql/RandomNumbers/knuthuniformrng.hpp File Reference 1061
12.373 ql/RandomNumbers/lecuyeruniformrng.cpp File Reference 1062
12.374 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference 1063
12.375 ql/RandomNumbers/mt19937uniformrng.cpp File Reference 1064
12.376 ql/RandomNumbers/mt19937uniformrng.hpp File Reference 1065
12.377 ql/RandomNumbers/randomarraygenerator.hpp File Reference 1066
12.378 ql/RandomNumbers/randomsequencegenerator.hpp File Reference 1067
12.379 ql/RandomNumbers/rngtypedefs.hpp File Reference 1068
12.380 ql/RandomNumbers/sobolrsg.cpp File Reference 1069
12.381 ql/RandomNumbers/sobolrsg.hpp File Reference 1070
12.382 ql/relinkablehandle.hpp File Reference 1071
12.383 ql/scheduler.cpp File Reference 1072
12.384 ql/scheduler.hpp File Reference 1073
12.385 ql/ShortRateModels/calibrationhelper.cpp File Reference 1074
12.386 ql/ShortRateModels/calibrationhelper.hpp File Reference 1075
12.387 ql/ShortRateModels/CalibrationHelpers/caphelper.cpp File Reference 1076
12.388 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference 1077
12.389 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.cpp File Reference . . . 1078
12.390 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference . . . 1079
12.391 ql/ShortRateModels/model.cpp File Reference 1080
12.392 ql/ShortRateModels/model.hpp File Reference 1081
12.393 ql/ShortRateModels/onefactormodel.cpp File Reference 1082
12.394 ql/ShortRateModels/onefactormodel.hpp File Reference 1083
12.395 ql/ShortRateModels/OneFactorModels/blackkarasinski.cpp File Reference 1084
12.396 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference . . . 1085
12.397 ql/ShortRateModels/OneFactorModels/coxingersollross.cpp File Reference . . . 1086
12.398 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference . . . 1087
12.399 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.cpp File Reference 1088
12.400 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference 1089
12.401 ql/ShortRateModels/OneFactorModels/hullwhite.cpp File Reference 1090
12.402 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference 1091
12.403 ql/ShortRateModels/OneFactorModels/vasicek.cpp File Reference 1092

12.404 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference 1093
12.405 ql/ShortRateModels/parameter.hpp File Reference 1094
12.406 ql/ShortRateModels/twofactormodel.cpp File Reference 1095
12.407 ql/ShortRateModels/twofactormodel.hpp File Reference 1096
12.408 ql/ShortRateModels/TwoFactorModels/g2.cpp File Reference 1097
12.409 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference 1098
12.410 ql/solver1d.hpp File Reference 1099
12.411 ql/Solvers1D/bisection.hpp File Reference 1100
12.412 ql/Solvers1D/brent.hpp File Reference 1101
12.413 ql/Solvers1D/falseposition.hpp File Reference 1102
12.414 ql/Solvers1D/newton.hpp File Reference 1103
12.415 ql/Solvers1D/newtonsafe.hpp File Reference 1104
12.416 ql/Solvers1D/ridder.hpp File Reference 1105
12.417 ql/Solvers1D/secant.hpp File Reference 1106
12.418 ql/swaptionvolstructure.hpp File Reference 1107
12.419 ql/termstructure.hpp File Reference 1108
12.420 ql/TermStructures/affinetermstructure.cpp File Reference 1109
12.421 ql/TermStructures/affinetermstructure.hpp File Reference 1110
12.422 ql/TermStructures/compoundforward.cpp File Reference 1111
12.423 ql/TermStructures/compoundforward.hpp File Reference 1112
12.424 ql/TermStructures/discountcurve.cpp File Reference 1113
12.425 ql/TermStructures/discountcurve.hpp File Reference 1114
12.426 ql/TermStructures/drifttermstructure.hpp File Reference 1115
12.427 ql/TermStructures/extendeddiscountcurve.cpp File Reference 1116
12.428 ql/TermStructures/extendeddiscountcurve.hpp File Reference 1117
12.429 ql/TermStructures/flatforward.hpp File Reference 1118
12.430 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference 1119
12.431 ql/TermStructures/impliedtermstructure.hpp File Reference 1120
12.432 ql/TermStructures/piecewiseflatforward.cpp File Reference 1121
12.433 ql/TermStructures/piecewiseflatforward.hpp File Reference 1122
12.434 ql/TermStructures/quantotermstructure.hpp File Reference 1123
12.435 ql/TermStructures/ratehelpers.cpp File Reference 1124
12.436 ql/TermStructures/ratehelpers.hpp File Reference 1125
12.437 ql/TermStructures/zerocurve.cpp File Reference 1126
12.438 ql/TermStructures/zerocurve.hpp File Reference 1127
12.439 ql/TermStructures/zerospreadedtermstructure.hpp File Reference 1128

12.440 ql/types.hpp File Reference 1129
12.441 ql/Utilities/combiningiterator.hpp File Reference 1130
12.442 ql/Utilities/couplingiterator.hpp File Reference 1131
12.443 ql/Utilities/filteringiterator.hpp File Reference 1132
12.444 ql/Utilities/iteratorcategories.hpp File Reference 1133
12.445 ql/Utilities/processingiterator.hpp File Reference 1134
12.446 ql/Utilities/steppingiterator.hpp File Reference 1135
12.447 ql/Volatilities/blackconstantvol.hpp File Reference 1136
12.448 ql/Volatilities/blackvariancecurve.cpp File Reference 1137
12.449 ql/Volatilities/blackvariancecurve.hpp File Reference 1138
12.450 ql/Volatilities/blackvariancesurface.cpp File Reference 1139
12.451 ql/Volatilities/blackvariancesurface.hpp File Reference 1140
12.452 ql/Volatilities/capflatvolvector.hpp File Reference 1141
12.453 ql/Volatilities/impliedvoltermstructure.hpp File Reference 1142
12.454 ql/Volatilities/localconstantvol.hpp File Reference 1143
12.455 ql/Volatilities/localvolcurve.hpp File Reference 1144
12.456 ql/Volatilities/localvolsurface.cpp File Reference 1145
12.457 ql/Volatilities/localvolsurface.hpp File Reference 1146
12.458 ql/Volatilities/swaptionvolmatrix.hpp File Reference 1147
12.459 ql/voltermstructure.cpp File Reference 1148
12.460 ql/voltermstructure.hpp File Reference 1149

13 QuantLib Example Documentation **1151**

13.1 AmericanOption.cpp 1151
13.2 BermudanSwaption.cpp 1156
13.3 DiscreteHedging.cpp 1161
13.4 EuropeanOption.cpp 1167
13.5 history_iterators.cpp 1179
13.6 swapvaluation.cpp 1180

14 Unstable features **1191**

15 Deprecated List **1193**

Part I

User Manual

Chapter 1

An introduction to QuantLib

1.1 Introduction

QuantLib (`http://quantlib.org/`) is a C++ library for financial quantitative analysts and developers.

QuantLib is Non-Copylefted Free Software released under the modified BSD License. It is also OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

QuantLib is free software and you are allowed to use, copy, modify, merge, publish, distribute, and/or sell copies of it under the conditions stated in the QuantLib **Copyright and License**(p. 24).

QuantLib and its documentation are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the **Copyright and License**(p. 24) for more details.

1.1.1 Disclaimer

At this time, this documentation is widely incomplete and must be regarded as a work in progress. Eventually, all QuantLib classes will be documented. However, no date is currently set for this goal. Join the mailing lists (`http://quantlib.org/mailinglists.html`) for the latest updates.

1.2 Project overview

The QuantLib project is at this time in *beta* status.

The following list is an overview of the goals set for release 1.0. It is open for suggestions, corrections, and additions which can be submitted through the `QuantLib-users` mailing list. Every proposal will be seriously considered.

1.2.1 Project goals

Interfaces

- yield term structure—done;
- financial instrument—done;
- deal—to do;
- folder/portfolio/book levels of deal aggregation (strict vertical inclusion)—to do;
- trade-group deal aggregation (horizontal cross section)—to do;
- volatility term structure for cap/floor and swaption—to do;

Generic Tools

- date/time module—done;
- one-dimensional solver—done;
- one-dimensional optimizer—to do;
- multi-dimensional solver—to do;
- multi-dimensional optimizer—in progress;
- linear algebra module—in progress;
- statistical module—in progress;
- PDE framework—in progress;
- Monte Carlo framework—in progress;

Financial Tools

- Black-Scholes analytic implementation—done;
- Black-Scholes PDE implementation (for American and exotic options)—done;
- Deposits, FRA, futures, swaps—partly done;
- Deposit/FRA/futures/swap yield term structure—partly done;
- Swaptions—in progress;
- Caps/floors—in progress;

- Bonds (IRR/duration)—to do;
- Treasury yield term structure (Nelson-Siegel/Vasichek-Fong)—to do;
- Short-rate models (BDT, BK, HW, etc.)—in progress;
- Local volatility models—in progress;
- Stochastic volatility—to do.

Assorted Possible Milestones

- Moving to the generic option pricing engine (see `http://quantlib.org/quep/quep005.html`) where each option can be priced with different engines: analytic, trees, Finite Differences, Monte Carlo. Allow everywhere for time varying volatilities and interest rates.
- Two dimensions FD framework, with satisfactory solution of the Free Boundary problem (early exercise).
- MC path-wise Greek evaluation.
- Full coverage of analytic models for option pricing.
- Complete coverage of basic Fixed Income instruments (cap/floor, swaption, Bermuda swaption) with all major one-factor models.
- Modeling of stochastic process with jumps, reversion, etc.
- Extraction of local volatilities surface from Black volatility surface.
- Implementation of FI products using FD framework.
- Low-discrepancy sequences for MC simulation.
- FD solution of weak path-dependent options (Asian, look-back, etc).
- Advanced MC variance reduction techniques.
- Implementation of Fixed Income products using MC framework.

1.3 Where to get QuantLib

1.3.1 QuantLib releases

Source code, documentation, modules, etc. of current and previous QuantLib releases can be downloaded from `http://quantlib.org/download.html`

Debian package is available from `http://ftp.us.debian.org/debian/pool/main/q/quantlib` (the .us can be replaced by .jp, .uk, .de, .it, etc.)

1.3.2 Current CVS snapshot

Instructions for anonymous CVS access are available at `http://quantlib.org/cvs.html`

Access to the CVS repository is intended mainly for developers and is not recommended to end users which should download the latest stable release instead.

1.4 Installation

1.4.1 Linux/Unix

The simplest way to compile and install QuantLib is:

1. get the sources from

   `http://quantlib.org/download.html` and untar them.

2. 'cd' to the QuantLib directory and type './configure' to configure the package for your system. If you're using 'csh' on an old version of System V, you might need to type 'sh ./configure' instead to prevent 'csh' from trying to execute 'configure' itself. Running 'configure' takes awhile. While running, it prints some messages telling which features it is checking for.

3. Type 'make' to compile the package.

4. Type 'make install' to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that 'configure' created (so you can compile the package for a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you will need some GNU tools that usually only developers use, and which are not required to build QuantLib from tarballs. These are automake, autoconf, libtool, GNU m4, GNU make, and maybe others. They all come with recent GNU/Linux distributions. To begin the build process after 'make distclean' start with 'sh ./bootstrap' which will prepare the package for compilation. You can then use 'configure' and 'make' in the usual way.

The file INSTALL.txt in the QuantLib source distribution contains more detailed instructions.

1.4.2 Win32

Binary installers are available at

`http://quantlib.org/download.html`

Visual C++ 6.0 projects files and Borland C++ makefiles are supplied in case one wants to rebuild the library.

The free Borland C++ compiler is available at

`http://www.borland.com/bcppbuilder/freecompiler/`

1.4.3 Macintosh

Codewarrior support is currently broken. QuantLib should compile under Mac OS X as outlined under **Linux/Unix**(p. **??**).

1.5 Usage

To use QuantLib classes in your own code just add

```
#include <ql/quantlib.hpp>
```

at the beginning of your source/header files.

Under the Examples folder you can find examples of QuantLib usage, including makefiles for gcc, Borland Free compiler, and Microsoft Visual C++. For the latter project files are also available.

1.5.1 Microsoft Visual C++

A few suggestions for Visual C++ users wanting to use QuantLib into their own application:

1. As long as you include ql/quantlib.hpp, you won't have to explicitly link your application to **QuantLib**(p. 107) library. This is automatically done by quantlib.hpp using the pragma statement:

```
#ifdef _DLL
    #ifdef _DEBUG
        #pragma comment(lib,"QuantLib_MTDLL_d.lib")
    #else
        #pragma comment(lib,"QuantLib_MTDLL.lib")
    #endif
#else
    #ifdef _DEBUG
        #pragma comment(lib,"QuantLib_d.lib")
    #else
        #pragma comment(lib,"QuantLib.lib")
    #endif
#endif
```

2. Your main() must be compiled with the same options that were used in compiling the QuantLib library - namely, you'll have to set the run-time library to "(Debug) Multithreaded (DLL)". For single-threaded version of **QuantLib**(p. 107) one must change the settings under project settings, "C/C++" tab, "Code Generation", and then re-build the library with the new settings. You'll have to check the "Use RTTI" option under the "C++ Language" category, too. Finally, you have to define NOMINMAX

3. To create your own project, add it to a new or existing Workspace. Under File | New | Projects select "Win32 Console Application" (or equivalent). Under Project | Settings select your project. Select settings for: "All configuration". On the "C/C++" tab select "Preprocessor" and under "Additional include directory" add "\$(QL_DIR)". This will add your current QuantLib installation directory to the include path. On the "Link" tab select "Input" and under "Additional library path" add "\$(QL_DIR)\lib\Win32\VisualStudio\". This will add your current QuantLib installation directory to the library path.

4. To compile in "Win32 Debug" configuration you will need the QuantLib debug libraries, available for download from `http://quantlib.org/download.html`

1.6 Supported platforms

The following is a list of configurations and compilers supported by QuantLib. All configuration names may have up to four parts: cpu-manufacturer-kernel-operating_system.

There are 3 officially supported configuration/compiler couples:

CONFIGURATION	COMPILER	COORDINATOR
i686-pc-linux-debian2.2	gcc 2.95 and 3.0.1	Luigi Ballabio
i686-pc-Win2000	MS Visual C++ 6	Ferdinando Ametrano
i686-pc-Win2000	Borland free compiler	Ametrano/Ballabio

Dirk Eddelbuettel is the maintainer of the QuantLib Debian packages. Liguo Song is the maintainer of the QuantLib RPM packages. QuantLib was reported to compile also under:

CONFIGURATION	COMPILER
FreeBSD	gcc 2.95.2
alpha-RedHat6.2	Compaq 6.3.9.3
i686-pc-cygwin	gcc 2.95.2
ppc-linux	gcc 2.95.2
sgi-linux	gcc 2.95
sparc-sun-solaris	gcc 2.95.2
arm-Debian	
hppa-Debian	
ia64-Debian	
m68k-Debian	
mips-Debian	
mipsel-Debian	
powerpc-Debian	
s390-Debian	
sparc-Debian	

For all these platforms Debian packages are available. If you compiled (or failed to compile) QuantLib on different configurations and/or compilers please report your feedback to: `quantlib-users@lists.sourceforge.net`

1.7 Version history

Release 0.3.4 - November 21th, 2003

MONTE CARLO FRAMEWORK

- MC European in one step with strike-independent vol curve (hopefully)
- Path pricer for Binary options. It should cover both European and American style options. Also known as: Digital, Binary, Cash-At-Hit, Cash-At-Expiry.
- Path pricers for barrier options

PRICING ENGINES FRAMEWORK

- More options moved to the new pricing engine framework: binary, barrier
- Changed setupEngine() into setupArguments(args)
- Moved pricing-engine machinery up to Instrument class

FIXED INCOME

- New basis-point sensitivity functions
- Added Swap::startDate() and maturity()
- Cap/floor fixing days taken into account

SHORT RATE MODELS

- An additional constraint can now be passed to the calibration

VOLATILITY FRAMEWORK

- Visitable volatility term structures

OPTIMIZATION FRAMEWORK

- Added composite constraint

PATTERNS

- Visitor, Alexandrescu-style (saves some code duplication)

MATH

- Added more integration algorithms contributed by Roman Gitlin
- Relaxed constaints on interval boundaries for integration algorithms
- Interpolation traits

TEST SUITE

- Added implied cap/floor term volatility test
- Added test for binary options in PricingEngine Framework.
- Added tests for Barrier options in PricingEngine Framework. Some Monte Carlo tests, but not comprehensive.

MISCELLANEA

- Conditionally allowed negative yields (disabled by default)
- Null calendar and simple day counter for reproducing theoretical calculations
- Fixed for VC++.Net compilation
- Added spec file for RPMs
- Added global flag for early/late payments
- Enabled test suite for Borland
- Removed OnTheEdge VC++ configurations
- Added VC++ configurations for static and dynamic Multithread libraries
- Upgraded to use Doxygen 1.3.4

Release 0.3.3 - September 3rd, 2003

MONTE CARLO FRAMEWORK

- Re-templatized Monte Carlo model based on traits.
- New path generator based on DiffusionProcess, TimeGrid, and externally initialized random number generator.
- Added Halton low discrepancy sequence.
- Added sequence generators: random sequence generator creates a sequence generator out of a random number generator. InvCumGaussianRsg creates a gaussian sequence generator out of a uniform (random or low discrepancy) sequence generator.
- RNG as constructor input constructor(long seed) deprecated.
- Mersenne Twister random number generator added
- Old PathPricers, PathGenerators, etc are available with a trailing _old
- Added Jäckel's Brownian Bridge (not used yet.)
- Sobol Random Sequence Generator. Unit and Jäckel.
- Added randomized Halton sequences.

FINITE DIFFERENCE FRAMEWORK

- Old class Grid no longer exists, use CenteredGrid to obtain the same result.

LATTICE FRAMEWORK

- Abstracted discretized option.
- Additive binomial trees. All binomial trees now use DiffusionProcess.
- Added Tian binomial tree.

PRICING ENGINES FRAMEWORK

- Partially implemented.
- Quanto forward compounded engines.
- Integral (european) pricing engine.

YIELD TERM STRUCTURE

- ZeroCurve: a term structure based on linear interpolation of zero yields.

FIXED INCOME

- Up-front and in-arrear indexed coupon.
- Specific implementation of compound forward rate from zero yield.
- Added compound forward and zero coupon implementations.
- Added Futures rate helper with specified maturity date.
- Added bucketed bps calculation.
- Added swap constructor using specified maturity date as well as added functionality in Scheduler.
- Added date-bucketed basis point sensitivity based on 1st derivative of zero coupon rate.

OPTIMIZATION FRAMEWORK

- Solvers now take any function. ObjectiveFunction disappeared.

PATTERNS

- Abstracted lazy object.
- Abstracted the curiously recurring template pattern.

DATE AND CALENDARS

- Added joint calendars.
- Tokyo, Stockholm, Johannesburg calendar improved.
- "MonthEndReference" business day rolling convention. Similar to "ModifiedFollowing", unless where original date is last business day of month all resulting dates will also be last business day of month.
- Added basic date generation starting from the end.

MATH

- Added Gauss-Kronrod integration algorithm.
- Added primitive polynomial modulo 2 up to dimension 18 (available up to dimension 27.)
- Added BicubicSplineInterpolation.
- Numerical Recipes algorithm is back since there is a problem with Nicolas' code: it is unable to fit a straight line, it waves around the line.
- Prime number generation.
- Acklam's approximation for inverse cumulative normal distribution function (replaced Moro's algorithm as default.)
- Added error function.
- Improved Cumulative Normal Distribution function using the error function.
- Matrix pseudo square algorithm using salvaging algorithm(s).
- Added SequenceStatistics.
- Major Statistic reworking.
- Added DiscrepancyStatistic that inherits from SequenceStatistic and extends it with the calculation of L2-discrepancy.
- HStatistics.
- Added first and second derivative ot cubic splines.

RISK MEASURES

- Introduced semiVariance and regret.
- Redefinition of average shorfall (normalization factor now is cumulative(target) instead of 1.0)

MISCELLANEA

- QuEP 9 "generic disposable objects" implemented.
- Added test suite.
- Dataformatters extended to format long integers, Ordinal numerals, power of two formatting.
- Exercise class adopted.
- Added user configuration section.
- Inhibited automatic conversion of Handle<T> to RelinkableHandle<T>.
- Diffusion process extended.
- Added strikeSensitivity to the Greeks.
- BS does handle t==0.0 and sigma==0.0.

- TimeGrid has been reworked.
- Added payoff file for Payoff classes. Added Cash-Or-Nothing and Asset-Or-Nothing payoff classes.
- Upgraded to use Doxygen 1.3.

Release 0.3.1 - February 4th, 2003

FINITE DIFFERENCE FRAMEWORK

- partially implemented QuEP 2 (`http://quantlib.org/quep.html`)

VOLATILITY FRAMEWORK

- added Black and local volatility interface

PRICING ENGINES FRAMEWORK

- partially implemented QuEP 5 (`http://quantlib.org/quep.html`)

YIELD TERM STRUCTURE

- interface revisited
- added discrete time forward methods
- added DiscountCurve (loglinear interpolated) and CompoundForward term structures
- ForwardSpreadedTermStructure moved under **QuantLib::TermStructures**(p. 148) namespace

FIXED INCOME

- Modified coupons so that the payment date can be after the end of the accrual period

MISCELLANEA

- added/verified holidays of many calendars
- added new calendars
- added new currencies
- more date formatters
- added Period(std::string&)
- it is now possible to advance a calandar using a Period
- added LogLinear Interpolation
- the allowExtrapolation boolean in interpolation classes has been removed from constructors and added to the operator()
- Renamed Solver1D::lowBound and hiBound
- bug fixes

BUILD PROCESS

- More autoconfiscated time functions and types
- Migrated to latest autotools
- added patches for Darwin and Solaris

Release 0.3.0 - May 6th, 2002

MONTE CARLO FRAMEWORK

- Path and MultiPath are time-aware
- McPricer: extended interface, improved convergency algorithm

FINITE DIFFERENCE FRAMEWORK

- added mixed (implicit/explicit) scheme, from which Crank-Nicolson, ImplicitEuler, and ExplicitEuler are now derived
- Finite Difference exercise conditions are now in the FiniteDifferences folder/namespace
- Finite Difference pricers now start with 'Fd' letters
- BSMNumericalOption became BsmFdOption

LATTICE FRAMEWORK

- introduced first version of the framework
- CRR and JR binomial trees

VOLATILITY FRAMEWORK

- early works on reorganization of vol structures

YIELD TERM STRUCTURE

- new TermStructure class based on affine model
- yield curves can be spreaded in term of zeros (ZeroSpreadedTermStructure) and forwards (ForwardSpreadedTermStructure)
- Added dates() and times() to PiecewiseFlatForward
- discount factor accuracy in the yield curve bootstrapping is an input
- added single factor short-rate models (Hull-White, Black-Karasinski)
- added two factor short-rate models framework
- cap/floor and swaption calibration helpers
- added bermudan swaption pricing example (including BK and HW calibrations)

FIXED INCOME

- cap/floor and swaption tree pricer
- cap/floor analytical pricer
- vanilla swaption Jamshidian pricer
- Added accruedAmount() to coupons
- Made cash flow vector builders into functions

OPTIMIZATION FRAMEWORK

- added conjugate gradient, simplex

PATTERNS

- implemented QuEP 8 and 10

MISCELLANEA

- added allowExtrapolation parameter to interpolaton classes
- added 2D bilinear interpolation
- better spline interpolation algorithm
- Added non-central chi-square distribution function.
- Improved Inverse Cumulative Normal Distribution using Moro's algorithm
- Introduced class representing stochastic processes
- added isExpired() to Instrument interface
- added functions folder and namespace for QuantLibXL and any other function-like interface to **QuantLib**(p. 107)
- Handle is now castable to an Handle of a compatible type
- added downsideVariance to the Statistics class
- kustosis() and skewness() now handles the case of stddev == 0 and/or variance == 0
- added Correlation Matrix to MultiVariateAccumulator
- enforced MS VC compilation settings
- added "-debug" to the QL_VERSION version string ifdef QL_DEBUG
- "make check" runs the example programs under Borland C++
- fixed compilation with "g++ -pedantic"
- Spread as market element
- new calendars introduced
- new Xibor Indexes introduced
- Added optional day count to libor indexes
- Shortened file names within 31 char limit to support HFS

Release 0.2.1 - December 3rd, 2001

MONTE CARLO FRAMEWORK

- Path and MultiPath are now classes on their own
- PathPricer now handles both Path and MultiPath
- MonteCarloModel now handles both single factor and multi factors simulations.
- McPricer now handles both single factor and multi factors pricing. New pricing interface
- antithetic variance-reduction technique made possible in Monte Carlo for both single factor and multi factors
- Control Variate specific class removed: control variation technique is now handled by the general MC model
- average price and average strike asian option refactored
- Sample as a (value,weight) struct
- random number generators moved under RandomNumbers folder and namespace

FINITE DIFFERENCE FRAMEWORK

- BackwardEuler and ForwardEuler renamed ImplicitEuler and ExplicitEuler, respectively
- refactoring of TridiagonalOperator and derived classes

YIELD TERM STRUCTURE AND FIXED INCOME

- Added some useful methods to term structure classes
- Allowed passing a quote to RateHelpers as double
- added FuturesRateHelpers (no convexity adjustment yet)
- PiecewiseFlatForward now observer of rates passed as MarketElements
- Unified Date and Time interface in TermStructure
- Added BPS to generic swap legs
- added term_structure+swap example
- Fixing days introduced for floating-coupon bond

PATTERNS

- Added factory pattern
- Calendar and DayCounter now use the Strategy pattern

VARIOUS

- used do-while-false idiom in QL_REQUIRE-like macros
- now using size_t where appropriate

- dividendYield is now a Spread instead of a Rate (that is: cost of carry is allowed)
- RelinkableHandle initialized with an optional Handle
- Worked around VC++ problems in History constructor
- added QL_VERSION and QL_HEX_VERSION
- generic bug fixes
- removed classes deprecated in 0.2.0

INSTALLATION FACILITIES

- improved and smoother Win32 binary installer

DOCUMENTATION

- general re-hauling
- improved and extended Monte Carlo documentation
- improved and extended examples
- Upgraded to Doxygen 1.2.11.1
- Added man pages for installed executables
- added docs in Windows Help format
- added info on "Win32 OnTheEdgeRelease" and "Win32 OnTheEdgeDebug" MS VC++ configurations
- additional information on how to create a MS VC++ project based on **QuantLib**(p. 107)

Release 0.2.0 - September 18th, 2001

- Library:
  - source code moved under ql, better GNU standards
  - gcc build dir can now be separated from source tree
  - gcc 3.0.1 port
  - clean compilation (no warnings)
  - bootstrap script on cygwin
  - Fixed automatic choice of seed for random number generators
  - Actual/actual classes
  - extended platform support (see table in documentation)
  - antithetic variance-reduction technique made possible in Monte Carlo
  - added dividend-Rho greek
  - First implementation of segment integral (to be redesigned)
  - Knuth random generator
  - Cash flows, scheduler, and swap (both generic and simple) added
  - added ICGaussian random generator

  - generic bug fixes
- Installation facilities:
  - improved and smoother Win32 binary installer
  - better distribution
  - debian packages available
- Documentation:
  - general re-hauling
  - added examples of using QuantLib and of projects based on QL

Release 0.1.9 - May 31st, 2001

- Library:
  - Style guidelines introduced (see `http://quantlib.org/style.html`) and partially enforced
  - full support for Microsoft Visual Studio
  - full support for Linux/gcc
  - momentarily broken support for Metrowerks CodeWarrior
  - autoconfiscation (with specialized config.*.hpp files for platforms without automake/autoconf support)
  - Include files moved under Include/ql folder and referenced as "ql/header.hpp"
  - Implemented expression templates techniques for array algebra optimization
  - Added custom iterators
  - Improved term structure
  - Added Asian, Bermudan, Shout, Cliquet, Himalaya, and Barrier options (all with greeks calculation, control variated where possible)
  - Added Helsinki and Wellington calendars
  - Improved Normal distribution related functions: cumulative, inverse cumulative, etc.
  - Added uniform and Gaussian random number generators
  - Added Statistics class (mean, variance, skewness, downside variance, etc.)
  - Added RiskMeasures class: VAR, average shortfall, expected shortfall, etc.
  - Added RiskStatistics class combining Statistics and RiskMeasures
  - Added sample accumulator for multivariate analysis
  - Added Monte Carlo tools
  - Added matrix-related functions (square root, symmetric Schur decomposition)
  - Added interpolation framework (linear and cubic spline interpolation implemented).
- Installation facilities:
  - Added Win32 GUI installer for binaries
- Documentation:
  - support for Doxygen 1.2.7
  - Added man documentation

Release 0.1.1 - November 21st, 2000

Initial release.

1.8 Todo List

Class AmericanCondition(p. 161) Unify the intrinsicValues/Payoff thing

Member Side(p. 224) Generalize for n-dimensional conditions

Class CADLibor(p. 234) check settlement days

Class CapFlatVolatilityVector(p. 246) Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Class CHFLibor(p. 255) check settlement days and day-count

Class ContinuousGeometricAPO(p. 272) add Average Strike version and make it backward starting

Class DirichletBC(p. 300) generalize to time-dependent conditions.

Class DiscreteGeometricAPO(p. 306) add analytical greeks

Class DiscreteGeometricASO(p. 307) add analytical greeks

Class ExplicitEuler(p. 335) add Richardson extrapolation

Class FdAmericanOption(p. 343) make american call with no dividends = european

Class FraRateHelper(p. 368) convexity adjustment should be implemented.

Class Interpolation2D(p. 429) Bicubic interpolation and bicubic spline

Class McDiscreteArithmeticAPO(p. 481) Continous Averaging version

Class McDiscreteArithmeticASO(p. 482) Continous Averaging version

Class MixedScheme(p. 500) add Douglas Scheme

Class MultiPath(p. 506) make it time-aware

Class NeumannBC(p. 511) generalize to time-dependent conditions.

Class ShoutCondition(p. 595) Unify the intrinsicValues/Payoff thing

Class SwaptionVolatilityMatrix(p. 629) Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Class TermStructure(p. 634) add derived class ParSwapTermStructure similar to ZeroYield-TermStructure, DiscountStructure, ForwardRateStructure

allow for different compounding rules and compounding frequencies

Class TimeGrid(p. 642) What was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Class ZARLibor(p. 678) check settlement days

Member FloatingRateCouponVector(p. 122)**(const Schedule &schedule, const std::vector< double > &nominals,** A suitable algorithm should be implemented for the calculation of the interpolated index fixing for a short/long first coupon.

1.9 Additional resources

The main QuantLib resource is the QuantLib site (`http://quantlib.org`).

Additional resources available from the above site include:

- current news;
- the QuantLib mailing lists (`http://quantlib.org/mailinglists.html`);
- the QuantLib FAQ (`http://quantlib.org/FAQ.html`);
- the QuantLib programming style guidelines (`http://quantlib.org/style.html`);
- a link to the QuantLib project page on SourceForge.net;
- links to pages for bug reports, patch submissions and feature requests;
- a page (`http://quantlib.org/extensions.html`) about how to use QuantLib in other languages/platforms;
- as well as links to additional quantitative finance resources.

1.10 The QuantLib Group

1.10.1 Authors

The QuantLib Group members are:

- Ferdinando Ametrano (nando AT ametrano DOT net), project manager
- Luigi Ballabio (luigi.ballabio@riskmap.net), library designer
- Nicolas Di Césaré, spline and optimizer
- Dirk Eddelbuettel: Debian maintainer, RQuantLib
- Neil Firth: Monte Carlo: Longstaff-Schwartz American option, Barrier option
- Roman Gitlin: integration algorithms
- Marco Marchioro (marco.marchioro@riskmap.net)
- Sadruddin Rejeb (sad@quantlib.org), interest rate models
- Niels Elken Sønderby: Gauss-Kronrod integration
- Liguo Song (Liguo.Song AT vanderbilt.edu): RPM maintainer

1.10.2 Contributors

- Mario Aleppo (mario.aleppo@riskmap.net)
- Toyin Akin
- James Battle
- Christopher Baus: inspiration for custom iterators
- Thomas Becker: inspiration for custom iterators
- Adolfo Benin (adolfo.benin@libero.it)
- David Binderman
- Jon Davidson - UK bank holidays fix
- Matteo Gallivanoni (matteo.gallivanoni@riskmap.net)
- Gilbert Peffer
- Peter Schmitteckert (quantlib@schmitteckert.com)
- Enrico Sirola (enrico.sirola@riskmap.net)
- Maxim Sokolov (maxim_sokolov@rambler.ru): examples
- Tomoya Kawanishi: Japan holidays fix
- David Schwartz: fixes for VC7
- Bernd Johannes Wuebben

1.11 Copyright and License

Copyright ©2002, 2003 Ferdinando Ametrano

Copyright ©2001, 2002, 2003 Nicolas Di Césaré

Copyright ©2003 Neil Firth Copyright ©2002, 2003 Roman Gitlin Copyright ©2001, 2002, 2003 Sadruddin Rejeb

Copyright ©2000, 2001, 2002, 2003 RiskMap srl

Copyright ©2003 Niels Elken Sønderby Copyright ©2003 StatPro Italia srl All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the QuantLib Group and its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.11.1 Comments on Copyright and License

QuantLib is Non-Copylefted Free Software [1] released under the modified BSD License [2] (also know as XFree86-style license).

QuantLib is Open Source [3] because of its license: it is OSI Certified Open Source Software [4]. OSI Certified is a certification mark of the Open Source Initiative [5].

The modified BSD License is GPL compatible as confirmed by the Free Software Foundation [6].

This license has been adopted to allow free use of QuantLib and its source, to make QuantLib flourish as a free-software/open-source project. It allows proprietary extensions to be commercialized.

[1] `http://www.gnu.org/philosophy/categories.html#Non-CopyleftedFreeSoftware`

[2] `http://www.opensource.org/licenses/bsd-license.html`

[3] http://www.opensource.org/docs/definition.html

[4] http://www.opensource.org/docs/certification_mark.html

[5] http://www.opensource.org

[6] http://www.gnu.org/philosophy/bsd.html

Chapter 2

QuantLib components

2.1 Core classes

Documentation for this part of the library is in progress.

The root namespace **QuantLib**(p. 107) contains what can be considered the QuantLib foundation, i.e., the core of abstract classes upon which the rest of the library is built.

The folder structure of the sources reflects the organization of classes and functions into namespaces. These are in turn chosen to represent definite *packages* or *class families*: the former being groups of classes forming a framework for particular applications, e.g., the PDE package for partial differential equations; and the latter being sets of classes which inherit from a common base class or otherwise share a common functionality and interface, e.g., the TermStructures or Operators families, respectively.

In the **QuantLib**(p. 107) namespace:

- A number of financial types is provided, such as **QuantLib::Rate**(p. 112), **QuantLib::Spread**(p. 112), or **QuantLib::DiscountFactor**(p. 112). All these types act as, and indeed are, doubles.
- The Date, Calendar and DayCounter classes are defined (see **Date and time calculations**(p. 28)).
- The abstract classes QuantLib::ForwardVolatilitySurface and QuantLib::SwaptionVolatilitySurface provide the common interface to concrete volatility models. No examples are available at this time, and the classes themselves will probably need some reworking.
- The abstract class **QuantLib::Solver1D**(p. 608) provides a common interface for the one-dimensional solvers implemented in the **QuantLib::Solvers1D**(p. 147) namespace.
- The abstract class **QuantLib::Index**(p. 417) provides the common interface of the classes implemented in the **QuantLib::Indexes**(p. 127) namespace.
- The abstract class **QuantLib::Instrument**(p. 422) provides a common interface and a calculation ramework for the classes implemented in the **QuantLib::Instruments**(p. 128) namespace.
- The abstract class **QuantLib::TermStructure**(p. 634) provides a common interface for the classes implemented in the **QuantLib::TermStructures**(p. 148) namespace.

2.2 Date and time calculations

Documentation for this part of the library is in progress.

2.2.1 Dates

The concrete class **QuantLib::Date**(p. 288) implements the concept of date. Its functionalities include:

- providing basic information such as weekday, day of the month, day of the year, month, and year;
- comparing two dates to determine whether they are equal, or which one is the earlier or later, or the difference between them expressed in days;
- incrementing or decrementing a date of a given number of days, or of a given period expressed in weeks, months, or years.

2.2.2 Calendars

The abstract class **QuantLib::Calendar**(p. 235) provides the interface for determining whether a date is a business day or a holiday for a given market, or incrementing/decrementing a date of a given number of business days. A number of calendars is contained in the **QuantLib::Calendars**(p. 117) namespace.

2.2.3 Day counters

The abstract class **QuantLib::DayCounter**(p. 292) provides more advanced means of measuring the distance between two dates according to a given market convention, both as number of days of fraction of year. A number of such conventions is contained in the **QuantLib::DayCounters**(p. 124) namespace, namely,

- Actual/360
- Actual/365
- Actual/Actual, which can be calculated according to:
  - ISMA and US Treasury convention, also known as "Actual/Actual (Bond)";
  - ISDA, also known as "Actual/Actual (Historical)";
  - AFB, also known as "Actual/Actual (Euro)";
- 30/360, which can be calculated according to:
  - USA convention;
  - European convention;
  - Italian convention.

2.3 Lattice methods

The framework corresponding to the QuantLib::Lattices namespace contains basic building blocks for pricing instruments using lattice methods (trees). A lattice, i.e. an instance of the abstract class **QuantLib::Lattices::Lattice**(p. 439), relies on one or several trees (each one approximating a diffusion process) to price an instance of the DiscretizedAsset class. Trees are instances of classes derived from **QuantLib::Lattices::Tree**(p. 648), classes which define the branching between nodes and transition probabilities.

2.3.1 Binomial trees

The binomial method is the simplest numerical method that can be used to price path-independent derivatives. It is usually the preferred lattice method under the Black-Scholes-Merton model. As an example, let's see the framework implemented in the **bsmlattice.hpp**(p. 845) file. It is a method based on a binomial tree, with constant short-rate (discounting). There are several approaches to build the underlying binomial tree, like Jarrow-Rudd or Cox-Ross-Rubinstein.

2.3.2 Trinomial trees

When the underlying stochastic process has a mean-reverting pattern, it is usually better to use a trinomial tree instead of a binomial tree. An example is implemented in the **QuantLib::Lattices::TrinomialTree**(p. 656) class, which is constructed using a diffusion process and a time-grid. The goal is to build a recombining trinomial tree that will discretize, at a finite set of times, the possible evolutions of a random variable y satisfying

$$dy_t = \mu(t, y_t)dt + \sigma(t, y_t)dW_t.$$

At each node, there is a probability $p_u, p_m and p_d$ to go through respectively the upper, the middle and the lower branch. These probabilities must satisfy

$$p_u y_{i+1,k+1} + p_m y_{i+1,k} + p_d y_{i+1,k-1} = E_{i,j}$$

and

$$p_u y^2_{i+1,k+1} + p_m y^2_{i+1,k} + p_d y^2_{i+1,k-1} = V^2_{i,j} + E^2_{i,j},$$

where k (the index of the node at the end of the middle branch) is the index of the node which is the nearest to the expected future value, $E_{i,j} = \mathbf{E}\left(y(t_{i+1})|y(t_i) = y_{i,j}\right)$ and $V^2_{i,j} = \mathbf{Var}\{y(t_{i+1})|y(t_i) = y_{i,j}\}$. If we suppose that the variance is only dependant on time $V_{i,j} = V_i$ and set y_{i+1} to $V_i\sqrt{3}$, we find that

$$p_u = \frac{1}{6} + \frac{(E_{i,j} - y_{i+1,k})^2}{6V_i^2} + \frac{E_{i,j} - y_{i+1,k}}{2\sqrt{3}V_i},$$

$$p_m = \frac{2}{3} - \frac{(E_{i,j} - y_{i+1,k})^2}{3V_i^2},$$

$$p_d = \frac{1}{6} + \frac{(E_{i,j} - y_{i+1,k})^2}{6V_i^2} - \frac{E_{i,j} - y_{i+1,k}}{2\sqrt{3}V_i}.$$

2.3.3 Bidimensional lattices

To come...

2.3.4 The QuantLib::DiscretizedAsset class

This class is a representation of the price of a derivative at a specific time. It is roughly an array of values, each value being associated to a state of the underlying stochastic variables. For the moment, it is only used when working with trees, but it should be quite easy to make a use of it in finite-differences methods. The two main points, when deriving classes from **QuantLib::DiscretizedAsset**(p. 308), are:

1. Define the initialisation procedure (e.g. terminal payoff for european stock options).
2. Define the method adjusting values, when necessary, at each time steps (e.g. apply the step condition for american or bermudan options). Some examples are found in QuantLib::Pricers::DiscretizedSwap and QuantLib::Pricers::DiscretizedSwaption.

2.4 The finite differences framework

This framework (corresponding to the **QuantLib::FiniteDifferences**(p. 125) namespace) contains basic building blocks for the numerical solution of a generic differential equation

$$\frac{\partial f}{\partial t} = Lf$$

where L is a differential operator in "space", i.e., one which does not contain partial derivatives in t but can otherwise contain any derivative in any other variable of the problem.

Writing the equation in the above form allows us to implement separately the discretization of the differential operator L and the time scheme used for the evolution of the solution. The **QuantLib::FiniteDifferences::FiniteDifferenceModel**(p. 352) class acts as a glue for such two steps—which are outlined in the following sections—and provides the interface of the resulting finite difference model for the end user. Furthermore, it provides the possibility of checking and operating on the solution array at each step—which is typically used to apply an exercise condition for an option. This is also outlined in a section below.

2.4.1 Differential operators

The discretization of the differential operator L depends on the discretization chosen for the solution f of the given equation.

Such choice is obvious in the 1-D case where the domain $[a, b]$ of the equation is discretized as a series of points $x_i, i = 0 \dots N-1$ (note that the index is zero based) where $x_i = a + hi$ and $h = (b-a)/(N-1)$. In turn, the solution f of the equation is discretized as an array $u_i, i = 0 \dots N-1$ whose elements are defined as $u_i = f(x_i)$. The discretization of the differential operator follows by substituting the derivatives with the corresponding incremental ratios defined in terms of the f_i. A number of basic operators are defined in the framework which can be composed to form more complex operators, namely:

the first derivative $\partial/\partial x$ is discretized as the operator D_+, defined as

$$D_+ u_i = \frac{u_{i+1} - u_i}{h}$$

and implemented in class **QuantLib::FiniteDifferences::DPlus**(p. 316); the operator D_-, defined as

$$D_- u_i = \frac{u_i - u_{i-1}}{h}$$

and implemented in class **QuantLib::FiniteDifferences::DMinus**(p. 314); and the operator D_0, defined as

$$D_0 u_i = \frac{u_{i+1} - u_{i-1}}{2h}$$

and implemented in class **QuantLib::FiniteDifferences::DZero**(p. 320). The discretization error of the above operators is $O(h)$ for D_+ and D_- and $O(h^2)$ for D_0;

the second derivative $\partial^2/\partial x^2$ is discretized as the operator $D_+ D_-$, defined as

$$D_+ D_- u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

and implemented in class **QuantLib::FiniteDifferences::DPlusDMinus**(p. 317). Its discretization error is $O(h^2)$.

The boundary condition for the above operators is by default linear extrapolation. Methods are currently provided for setting other kinds of boundary conditions to a tridiagonal operator which

these operators inherit, namely, Dirichlet—i.e., constant value—and Neumann—i.e., constant derivative—boundary conditions. This might change in the future as boundary conditions could be astracted and passed as an additional argument to the model.

A programmer can also implement its own operator. However, in order to fit into this framework it will have to implement a required interface depending on the chosen evolver (see below). Also, it is currently required to manage itself any boundary conditions. Again, this could change in the future.

On the other hand, there is no obvious choice in the 2-D case. While it is immediate to discretize the domain into a series of points (x_i, y_j) and the solution into a matrix $f_{ij} = f(x_i, y_j)$, there is a number of ways into which the f_{ij} can be arranged into an array—each of them determining a different discretization of the differential operators. One of such ways was implemented in the LexicographicalView class, while others will be implemented in the future. No 2-D operator is currently implemented.

2.4.2 Time schemes

Once the differential operator L has been discretized, a number of choices are available for discretizing the time derivative at the left-hand side of the equation.

In this framework, such choice is encapsulated in so-called evolvers which, given L and the solution $u^{(k)}$ at time t_k, yield the solution $u^{(k-1)}$ at the previous time step.

A number of evolvers are currently provided in the library which implement well-known schemes, namely,

the forward Euler explicit scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = Lu^{(k)}$$

hence

$$u^{(k-1)} = (I - \Delta t L)\, u^{(k)}$$

from which $u^{(k-1)}$ can be obtained directly;

the backward Euler implicit scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = Lu^{(k-1)}$$

hence

$$(I + \Delta t L)\, u^{(k-1)} = u^{(k)}$$

from which $u^{(k-1)}$ can be obtained by solving a linear system;

the Crank-Nicolson scheme in which the equation is discretized as

$$\frac{u^{(k)} - u^{(k-1)}}{\Delta t} = L\frac{u^{(k)} + u^{(k-1)}}{2}$$

hence

$$\left(I + \frac{\Delta t}{2} L\right) u^{(k-1)} = \left(I - \frac{\Delta t}{2} L\right) u^{(k)}$$

from which $u^{(k-1)}$ can be obtained by solving a linear system.

Each of the above evolvers forces a set of interface requirements upon the differential operator which are detailed in the documentation of the corresponding class, namely, **QuantLib::FiniteDifferences::ExplicitEuler**(p. 335), **QuantLib::FiniteDifferences::ImplicitEuler**(p. 408), and **QuantLib::FiniteDifferences::CrankNicolson**(p. 282), respectively.

A programmer could implement its own evolver, which does not need to inherit from any base class.

However, it must implement the following interface:

```
class Evolver {
  public:
    typedef ... arrayType;
    typedef ... operatorType;
    // constructors
    Evolver(const operatorType& D);
    // member functions
    void step(arrayType& a, Time t) const;
    void setStep(Time dt);
};
```

Finally, we note again that the pricing of an option requires the finite difference model to solve the corresponding equation *backwards* in time. Therefore, given a discretization u of the solution at a given time t, the call

```
evolver.step(u,t)
```

must calculate the discrete solution at the *previous* time, $t - dt$.

2.4.3 Step conditions

A finite difference model can be passed a step condition to be applied at each step during the rollback of the solution (e.g. the early exercise American condition). Such condition must be embodied in a class derived from **QuantLib::FiniteDifferences::StepCondition**(p. 612) and must implement the interface of the latter, namely,

```
class MyCondition : public StepCondition<arrayType> {
  public:
    void applyTo(arrayType& a, Time t) const;
};
```

2.4.4 An example of finite difference model

The Black-Scholes equation can be written in the above form as

$$\frac{\partial f}{\partial t} = -\frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2} - \nu\frac{\partial f}{\partial x} + rf.$$

It can be seen that the operator L_{BS} is

$$L_{BS} = -\frac{\sigma^2}{2}\frac{\partial^2}{\partial x^2} - \nu\frac{\partial}{\partial x} + rI$$

and can be built from the basic operators provided in the library as

$$L_{BS} = -\frac{\sigma^2}{2}D_+D_- - \nu D_0 + rI.$$

Its implementation closely reflects the above decomposition and can be written as

```
class BlackScholesOperator : public TridiagonalOperator {
  public:
    BlackScholesOperator(
        double sigma, double nu,    // parameters of the
        Rate r,                     // Black-Scholes equation;
        unsigned int points,        // number of discretized points;
        double h)                   // grid spacing.
    : TridiagonalOperator(
        // build the operator by adding basic ones
        - (sigma*sigma/2.0) * DPlusDMinus(points,h)
        - nu * DZero(points,h)
        + r * TridiagonalOperator::identity(points)
    ) {}
};
```

taking as inputs the relevant parameters of the equation (σ, ν and r) as well as model parameters such as the number N of grid points and their spacing h.

As simple example cases, we will use the above operator to price both an European and an American option. The parameters of the two options will be the same, namely, they will be both call options with underlying price $u = 100$, strike $s = 95$, residual time $T = 1$ year, dividend yield $q = 3\%$ and volatility $\sigma = 10\%$. The risk-free rate will be $r = 5\%$. Such parameters are expressed using QuantLib types as

```
Option::Type type = Option::Call;
double underlying = 100.0, strike = 95.0;
Time residualTime = 1.0;
Rate dividendYield = 0.03, riskFreeRate = 0.05;
double volatility = 0.10;
```

The grid upon which the model will act will be a logarithmic grid of underlying prices, i.e., f will be defined in a range $[\ln u_{min}, \ln u_{max}]$ discretized as an array $x_i, i = 0 \ldots N-1$ with $x_i = \ln u_{min} + ih$ and $h = (\ln u_{max} - \ln u_{min})/(N-1)$. Such a grid and the corresponding vector of actual prices can be built as shown in the code below. The domain of the model will be defined as $[\ln u - \Delta, \ln u + \Delta]$ where $\Delta = 4\sigma\sqrt{T}$. A number of grid points $N = 101$ will be used.

```
unsigned int gridPoints = 101;
Array grid(gridPoints), prices(gridPoints);
double x0 = QL_LOG(underlying);
double Delta = 4.0*volatility*QL_SQRT(residualTime);
double xMin = x0 - Delta, xMax = x0 + Delta;
double h = (xMax-xMin)/(gridPoints-1);
for (unsigned int i=0; i<gridPoints; i++) {
    grid[i] = xMin + i*h;
    prices[i] = QL_EXP(grid[i]);
}
```

The initial condition is determined by the values of the option at maturity, i.e., either the difference between underlying price and strike if such difference is positive, or 0 if that is not the case (the above will have to be suitably modified for a put option or a straddle.) Such "initial" condition will be rolled back in time by our model.

```
Array exercisingValue(gridPoints);
for (unsigned int i=0; i<gridPoints; i++)
    exercisingValue[i] = QL_MAX(prices[i]-strike,0.0);
```

Now the differential operator can be initialized. Also, Neumann initial conditions are set which correspond to the initial value of the derivatives at the boundaries (see the BoundaryCondition class documentation for details).

```
double nu = riskFreeRate - dividendYield - volatility*volatility/2.0;
TridiagonalOperator L = BlackScholesOperator(volatility, nu,
    riskFreeRate, gridPoints, h);
L.setLowerBC(BoundaryCondition(BoundaryCondition::Neumann,
    exercisingValue[1]-exercisingValue[0]));
L.setUpperBC(BoundaryCondition(BoundaryCondition::Neumann,
    exercisingValue[gridPoints_-1]-exercisingValue[gridPoints_-2]));
```

We are now already set for the pricing of the European option. Also, the exercise condition is the only thing still to be defined for the American option to be priced. Such condition is equivalent to the statement that at each time step, the value of the option is the maximum between the profit realized in exercising the option (which we already calculated and stored in `exercisingValue`) and the value of the option should we keep it (which corresponds to the solution rolled back to the current time step). This logic can be implemented as:

```
class ExerciseCondition : public StepCondition<Array> {
  public:
    ExerciseCondition(const Array& exercisingValue)
    : exercisingValue_(exercisingValue) {}
    void applyTo(Array& a, Time) const {
        for (unsigned int i = 0; i < a.size(); i++)
            a[i] = QL_MAX(a[i], exercisingValue_[i]);
    }
  private:
    Array exercisingValue_;
};
```

Everything is now ready. The model can be created gluing the piece together by means of the **QuantLib::FiniteDifferences::FiniteDifferenceModel**(p. 352) class. The current value of the option is calculated by rolling back the solution to the current time, i.e., $t = 0$, and by taking the value corresponding at the current underlying price—which by construction corresponds to the central value provided that the number of grid points is odd.

```
unsigned int timeSteps = 365;

// build the model - Crank-Nicolson scheme chosen
FiniteDifferenceModel<CrankNicolson<TridiagonalOperator> > model(L);

// European option
Array f = exercisingValue;  // initial condition
model.rollback(f, residualTime, 0.0, timeSteps);
double europeanValue = valueAtCenter(f);

// American option
f = exercisingValue;        // reset
Handle<StepCondition<Array> > condition(
    new ExerciseCondition(exercisingValue));
model.rollback(f, residualTime, 0.0, timeSteps, condition);
double americanValue = valueAtCenter(f);
```

2.5 The Monte Carlo framework

> Anyone attempting to generate random numbers by deterministic means is, of course, living in a state of sin. — John Von Neumann

Warning: this section of the documentation is currently outdated.

This framework (corresponding to the **QuantLib::MonteCarlo**(p. 135) namespace) contains basic building blocks for the numerical calculation of the integral

$$\int_{\Omega} f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

where $p(\mathbf{x})$ is a normalized probability function. Monte Carlo methods solve the above integral by approximating it with the discrete sum

$$\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)w(\mathbf{x}_i)$$

where the $\mathbf{x}_i$ are drawn from $p(\mathbf{x})$, possibly with a weight $w(\mathbf{x}_i)$ — which otherwise can be considered uniformly equal to 1.

The above sum has a straightforward interpretation in the case of a derivative product, namely, the $\mathbf{x}_i$ are N generated random paths which the value of the underlying can possibly follow, while the $f(\mathbf{x}_i)$ are the values of the derivative on each of such paths. The sum above can therefore be taken as an estimate of the price of the derivative, namely, the average of its value on all possible paths — or rather all considered paths. Such a method enables the user to construct pricing classes for an unlimited range of derivatives, most notably path-dependent ones which cannot be priced by means of finite difference methods.

It must also be mentioned that for all such methods, the error e on the estimated value is proportional to the square root of the number of samples N. A number of so-called *variance-reduction* methods have been found which allows one to reduce the coefficient of proportionality between e and $1/\sqrt{N}$.

Separate implementations are provided in the library for the three components of the above average, namely, the random drawing of the $\mathbf{x}_i$, the evaluation of the $f(\mathbf{x}_i)$, and the averaging process itself. The **QuantLib::MonteCarlo::MonteCarloModel**(p. 504) class acts as a glue for such three steps — which are outlined in the following sections — and provides the interface of the resulting Monte Carlo model to the end user.

2.5.1 Path generation

The Black-Scholes equation

$$\frac{\partial f}{\partial t} + \frac{\sigma^2}{2}\frac{\partial^2 f}{\partial x^2} + \nu\frac{\partial f}{\partial x} - rf = 0,$$

where r is the risk-free rate, σ is the volatility of the underlying, and $\nu = r - \sigma^2/2$, has the form of a diffusion process. According to this heuristic interpretation **(1)**(p. **??**), paths followed by the logarithm of the underlying would be Brownian random walks with a constant drift ν per unit time and a standard deviation $\sigma\sqrt{T}$ over a time T.

Therefore, the paths to be generated for a Monte Carlo model of the Black-Scholes equation will be vectors of successive variations of the logarithm of the underlying price over M consecutive time intervals $\Delta t_i, i = 0 \ldots M-1$. Each such variation will be drawn from a Gaussian distribution

with average $\nu\Delta T_i$ and standard deviation $\sigma\sqrt{\Delta T_i}$ — or possibly $\nu_i\Delta T_i$ and $\sigma_i\sqrt{\Delta T_i}$ should ν and σ vary in time.

The **QuantLib::MonteCarlo::Path**(p. 543) class stores the variation vector decomposed in its drift (determined) and diffusion (random) components. As shown below, this allows the implementation of antithetic variance reduction techniques.

The **QuantLib::MonteCarlo::MultiPath**(p. 506) class is a straightforward extension which acts as a vector of Path objects.

Classes are provided which generate paths and multi-paths with the desired drift and diffusion components, namely, **QuantLib::MonteCarlo::PathGenerator**(p. 544) and **QuantLib::MonteCarlo::MultiPathGenerator**(p. 507).

For the time being, the path generator is initialized with a constant drift and variance. This requirement will most likely be relaxed in the next release. The multi-path generator is initialized with an array of constant drifts—one for each single asset—and a covariance matrix which encapsulates the relations between the diffusion components of the single assets.

The time discretization of the (multi)paths can be specified either as a given number of equal time steps over a given time span, or as a vector of explicitly specified times at which the path will be sampled.

2.5.2 Pricing an instrument on a path

The **QuantLib::MonteCarlo::PathPricer**(p. 546) class is the base class from which path pricers must inherit. The only method which subclasses are required to implement is

```
double operator()(const P&) const;
```

where `P` can be Path or MultiPath depending on the derivative whose value must be calculated.

Similarly, the term *path* will be used in the following discussion as meaning either path or multi-path depending on the context. The term *single path* is not to be taken as opposite to multi-path, but rather as meaning "a single instance of a (multi)path" as opposed to the set of all generated (multi)paths.

The above method encapsulates the pricing of the derivative on a single path and must return its value had the evolution of the underlying(s) followed the path passed as argument. For this reason, control variate techniques (see below) must not be implemented at this level since they would cause the returned value to differ from the actual price of the derivative on the path.

Instead, antithetic variance-reduction techniques can be effectively implemented at this level and indeed are used in the pricers currently included in the library.

In short, such techniques consist in pricing an option on both the given path and its antithetic, the latter being a path with the same drift and the opposite diffusion component. The value of the sample is defined as the average of the prices on the two paths.

A generic implementation of antithetic techniques could consist of a path pricer class which takes a concrete path pricer upon construction and whose operator() simply proxies two calls to the contained pricer, passing the given path and its antithetic, and averages the result. However, this would not take full advantage of the technique.

In fact, it must be noted that using antithetic paths not only reduces the variance *per se* but also allows to factor out calculations commons to a path and its antithetic, thus reducing greatly the computation time. Therefore, such techniques are best implemented inside the path pricer itself, whose algorithm can fully exploit such factorization.

A number of path pricers are available in the library and are listed in the **QuantLib::MonteCarlo**(p. 135) namespace reference.

2.5.3 Accumulating and averaging samples

The class **QuantLib::MonteCarlo::MonteCarloModel**(p. 504) encapsulates the general structure of a Monte Carlo calculations, namely, the generation of a number of paths, the pricing of the derivative on each path, and the averaging of the results to yield the actual derivative price.

As outlined above, the first two steps are delegated to a path generator and a path pricer. The third step is also delegated to an object which accumulates weighted values and returns the statistic properties of the set of such values. One such class provided by the library is **QuantLib::Math::Statistics**(p. 133).

The concern of the Monte Carlo model is therefore to act as a glue between such three components and can be expressed by the following pseudo-code:

```
given pathGenerator, pathPricer, accumulator;
for i in number of samples {
    path,weight = pathGenerator.next();
    price = pathPricer(path);
    accumulator.add(price,weight);
}
```

The Monte Carlo model also provides the user with the possibility to take advantage of control-variate techniques.

Such techniques consist in pricing a portfolio from which the price of the derivative can be deduced, but with a lower variance than the derivative alone.

In our current implementation, static-hedge control variate is used, namely, the formed portfolio is long of the derivative we need to price and short of a similar derivative whose price can be calculated analytically. The value of the portfolio on a given path will of course be given by the difference of the values of the two derivatives on such path. However, due to the similarity between the derivatives, the portfolio price will have a lower variance than either derivative alone since any variation in the price of the latter will be partly compensated by a similar variation in the price of the other. Lastly, given the portfolio price, the price of the derivative we are interested in can be deduced by adding the analytic value of the other.

In order to use such technique, the user must provide the model with a path pricer for the additional option and the value of the latter. The action of the Monte Carlo model is in this case expressed as:

```
given pathGenerator, pathPricer, cvPathPricer, cvPrice, accumulator;
for i in number of samples {
    path,weight  = pathGenerator.next();
    portfolioPrice = pathPricer(path) - cvPathPricer(path);
    accumulator.add(portfolioPrice+cvPrice,weight);
}
```

Martingale (a.k.a. dynamic-hedge) control variate techniques are planned for future releases.

A **QuantLib::Pricers::McPricer**(p. 490) class is also available which wraps the typical usage of a Monte Carlo model.

Details on the Monte Carlo Pricer interface will be available in the **Pricers**(p. **??**) section.

2.5.4 Examples of Monte Carlo models

As a simple example, we will use the outlined tools to price an European option by means of Monte Carlo techniques.

Given a current underlying price u_0 and a path $p = [p_1, \ldots, p_N]$ where every variation p_i is the sum of a drift term d_i and a random diffusion term r_i, the price of the underlying at maturity is

$$u = u_0 \prod_1^N e^{p_i} = u_0 \exp\left(\sum_1^N p_i\right) = u_0 \exp\left(\sum_1^N d_i + \sum_1^N r_i\right)$$

while the price on the antithetic path — i.e., same drift and opposite diffusion — is

$$u_0 \exp\left(\sum_1^N d_i - \sum_1^N r_i\right).$$

The corresponding path pricer can be implemented as:

```
class EuropeanPathPricer : public PathPricer<Path> {
  public:
    EuropeanPathPricer(Option::Type type, double underlying,
                       double strike, DiscountFactor discount,
                       bool useAntithetic)
    // just store the needed parameters
    : type_(type), underlying_(underlying), strike_(strike),
      discount_(discount), useAntithetic_(useAntithetic) {}
    // here is the logic
    double operator()(const Path& path) const {

        size_t n = path.size();

        // factor out the sums in the formula above
        double sum_d = 0.0, sum_r = 0.0;
        for (size_t i = 0; i < n; i++) {
            sum_d += path.drift()[i];
            sum_r += path.diffusion()[i];
        }

        // calculate final underlying price on path
        double price = underlying_*QL_EXP(sum_d+sum_r);

        // calculate payoff
        double payoff;
        switch (type_) {
          case Option::Call;
            payoff = QL_MAX(price-strike,0.0);
            break;
          // other cases are left as an exercise to the reader
          ...
        }

        // current value of the option is the discounted payoff
        double optionValue = payoff*discount_;

        // stop here if not antithetic...
        if (!useAntithetic_)
            return optionValue;

        // ...otherwise calculate the value on the antithetic path
        double antiPrice = underlying_*QL_EXP(sum_d-sum_r);

        // calculate payoff and option value as above
        ...
```

```
        // return the average of the results on the two paths
        return (optionValue + antiOptionValue)/2.0;
    }
  private:
    // stored parameters
    ...
};
```

The path pricer can now be used in a model. Let us assume the following parameters:

```
Option::Type type = Option::Call;
double underlying = 100.0, strike = 95.0;
Time residualTime = 1.0;
Rate dividendYield = 0.03, riskFreeRate = 0.05;
double volatility = 0.10;
```

The path generator can be instantiated as

```
// parameters of the Black-Scholes equation
double vol2 = volatility*volatility;
double nu = riskFreeRate - dividendYield - vol2/2.0;
// in this case we are only interested in the final underlying price.
// Therefore, we can cover all the residual time in one big time step.
int timeSteps = 1;

Handle<GaussianPathGenerator> pathGenerator(
    new GaussianPathGenerator(nu,vol2,residualTime,timeSteps));
```

where QuantLib::MonteCarlo::GaussianPathGenerator is a typedef to a path generator using the default choice for a Gaussian random number generator.

The path pricer is instantiated as

```
// discount at maturity
DiscountFactor discount = QL_EXP(-riskFreeRate*residualTime);
bool antithetic = true;

Handle<PathPricer<Path> > pathPricer(
    new EuropeanPathPricer(type,underlying,strike,discount,antithetic));
```

The model can now be created and used as following:

```
// number of samples to be generated
size_t samples = 1000000;

// pass the path generator and pricer we just created and a
// newly instantiated Statistics object
MonteCarloModel<Statistics,GaussianPathGenerator,PathPricer> model(
    pathGenerator,pathPricer,Statistics());

model.addSamples(samples);

// now get the results: the option price is given by value with
// a confidence level given by error
value = model.sampleAccumulator().mean();
error = model.sampleAccumulator().errorEstimate();
```

More examples of path pricers can be found in the **QuantLib::MonteCarlo**(p. 135) namespace, while examples of more sophisticated pricers which uses them in Monte Carlo models can be found in the **QuantLib::Pricers**(p. 139) namespace.

2.5.5 Notes

(1)(p. **??**) A more rigorous approach would lead us to integrate the above equation by means of Green functions or Laplace transforms. Both such methods would show that the price at time $t = 0$ of an option with payoff $G(S(T))$ where $S(T)$ is the underlying price at expiry is given by the integral

$$\int_{-\infty}^{\infty} e^{-rT} G(S_0 e^{\xi}) \frac{1}{\sqrt{2\pi\sigma^2 T}} \exp\left(-\frac{(\xi - \nu T)^2}{2\sigma^2 T}\right) \mathrm{d}\xi$$

where S_0 is the price of the underlying at $t = 0$. It can be seen that the above integral is of the form shown at the beginning of this section, namely, the pricing function is

$$f(x) = e^{-rT} G(S_0 e^{x})$$

and can be interpreted as the option payoff discounted to the present time, while the probability distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2 T}} \exp\left(-\frac{(x - \nu T)^2}{2\sigma^2 T}\right).$$

which again shows that the logarithms of the underlying prices at time T are distributed as a Gaussian with average νT and standard deviation $\sigma\sqrt{T}$.

2.6 The short-rate modelling framework

This framework (corresponding to the ShortRateModels namespace) implements some single-factor and two-factor short rate models. The models implemented in this library are widely used by practitionners. For the moment, the ShortRateModels::Model class defines the short-rate dynamics with stochastic equations of the type

$$dx_i = \mu(t, x_i)dt + \sigma(t, x_i)dW_t$$

where $r = f(t, x)$. If the model is affine (i.e. derived from the **QuantLib::ShortRateModels::AffineModel**(p. 158) class), analytical formulas for discount bonds and discount bond options are given (useful for calibration).

2.6.1 Single-factor models

The Hull & White model

$$dr_t = (\theta(t) - \alpha(t)r_t)dt + \sigma(t)dW_t$$

When α and σ are constants, this model has analytical formulas for discount bonds and discount bond options.

The Black-Karasinski model

$$d\ln r_t = (\theta(t) - \alpha \ln r_t)dt + \sigma dW_t$$

No analytical tractability here.

The extended Cox-Ingersoll-Ross model

$$dr_t = (\theta(t) - kr_t)dt + \sigma\sqrt{r_t}dW_t$$

There are analytical formulas for discount bonds (and soon for discount bond options).

2.6.2 Calibration

The class CalibrationHelper is a base class that facilitates the instanciation of market instruments used for calibration. It has a method marketValue() that gives the market price using a Black formula, and a modelValue() method that gives the price according to a model

Derived classed are QuantLib::ShortRateModels::CalibrationHelpers::CapHelper and QuantLib::ShortRateModels::CalibrationHelpers::SwaptionHelper.

For the calibration itself, you must choose an optimization method that will find constant parameters such that the value:

$$V = \sqrt{\sum_{i=1}^{n} \frac{(T_i - M_i)^2}{M_i}},$$

where T_i is the price given by the model and M_i is the market price, is minimized. A few optimization methods are available in the QuantLib::Optimization namespace, and more are on the way...

2.6.3 Two-factor models

2.6.4 Pricers

Analytical pricers

If the model is affine, i.e. discount bond options formulas exist, caps are easily priced since they are a portfolio of discount bond options. Such a pricer is implemented in **QuantLib::Pricers::AnalyticalCapFloor**(p. 164). In the case of single-factor affine models, swaptions can be priced using the Jamshidian decomposition, implemented in **QuantLib::Pricers::JamshidianSwaption**(p. 432).

Using Finite Differences

(Doesn't work for the moment) For the moment, this is only available for single-factor affine models. If $x = x(t, r)$ is the state variable and follows this stochastic process:

$$dx_t = \mu(t, x)dt + \sigma(t, x)dW_t$$

any european-style instrument will follow the following PDE:

$$\frac{\partial P}{\partial t} + \mu\frac{\partial P}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 P}{\partial x^2} = r(t, x)P$$

The adequate operator to feed a Finite Difference Model instance is defined in the **QuantLib::FiniteDifferences::OneFactorOperator**(p. 532) class.

Using Trees

Each model derived from the single-factor model class has the ability to return a trinomial tree. For yield-curve consistent models, the fitting parameter can be determined either analytically (when possible) or numerically. When a tree is built, it is then pretty straightforward to implement a pricer for any path-independant derivative. Just implement a class derived from NumericalDerivative (see QuantLib::Pricers::NumericalSwaption for example) and roll it back until the present time... Just look at **QuantLib::Pricers::TreeCapFloor**(p. 649) and **QuantLib::Pricers::TreeSwaption**(p. 650) for working pricers.

2.7 Currencies and FX rates

Documentation for this part of the library is in progress.

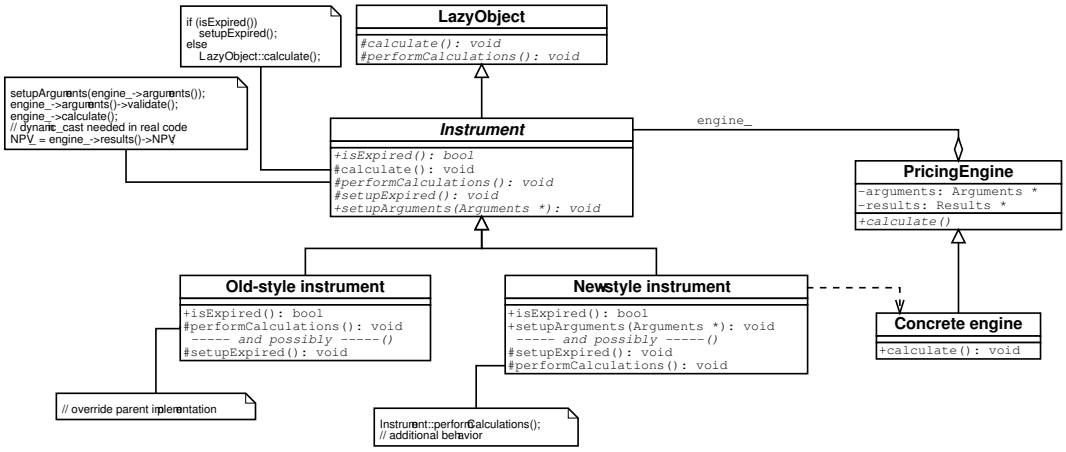
2.8 Instruments and pricers

Documentation for this part of the library is in progress.

This section will include documentation for classes such as **QuantLib::Instrument**(p. 422), its derived classes in the **QuantLib::Instruments**(p. 128) namespace, and the pricing engines in the **QuantLib::Pricers**(p. 139) namespace.

2.8.1 Instruments and pricers

Since version 0.3.4, the Instrument class was reworked as shown in the following figure.



On the one hand, the checking of the expiration condition is now performed in a method `isExpired()` separated from the actual calculation, and a `setupExpired()` method is provided. The latter sets the NPV to 0.0 and can be extended in derived classes should any other results be returned.

On the other hand, the pricing-engine machinery previously contained in the Option class was moved upwards to the Instrument class. Also, the `setupEngine()` method was replaced by a `setupArguments(Arguments*)` method. This allows one to cleanly implement containment of instruments with code such as:

```
class FooArguments : public Arguments { ... };

class Foo : public Instrument {
  public:
    void setupArguments(Arguments*);
    ...
};

class FooOptionArguments : public FooArguments { ... };

class FooOption : public Option {
  private:
    Foo underlying_;
  public:
    void setupArguments(Arguments* args) {
        underlying_.setupArguments(args);
        // set the option-specific part
    }
    ...
};
```

which was more difficult to write with `setupEngine()`.

Therefore, there are now two ways to inherit from Instrument, namely:

1. implement the `isExpired` method, and completely override the `performCalculations` method so that it bypasses the pricing-engine machinery. If the class declared any other results beside `NPV_` and `errorEstimate_`, the `setupExpired` method should also be extended so that those results are set to a value suitable for an expired instrument. This was the migration path taken for all instruments not previously deriving from the Option class.
2. define suitable argument and result classes for the instrument and implement the `isExpired` and `setupArguments` methods, reusing the pricing-engine machinery provided by the default `performCalculations` method. The latter can be extended by first calling the default implementation and then performing any additional tasks required by the instrument—most often, copying additional results from the pricing engine results to the corresponding data members of the instrument. As in the previous case, the `setupExpired` method can be extended to account for such extra data members.

2.9 Math tools

Documentation for this part of the library is in progress.

Math facilities of the library include:

2.9.1 Random number and low discrepancy sequence generators

Documentation for this part of the library is in progress.

Implementations of random number and low discrepancy sequence generators. They share the **QuantLib::RandomNumbers**(p. 142) namespace.

2.9.2 One-dimensional solvers

Documentation for this part of the library is in progress.

The abstract class **QuantLib::Solver1D**(p. 608) provides the interface for one-dimensional solvers which can find the zeroes of a given function.

A number of such solvers is contained in the **QuantLib::Solvers1D**(p. 147) namespace.

The implementation of the algorithms was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery - Chapter 9

Some work is needed to resolve the ambiguity of the root finding accuracy defition: for some algorithms it is the x-accuracy, for others it is f(x)-accuracy.

2.9.3 Optimizers

The optimization framework (corresponding to the Optimization namespace) implements some multi-dimensional minimizing methods. The function to be minimized is to be derived from the **QuantLib::Optimization::CostFunction**(p. 273) base class (if the gradient is not analytically implemented, it will be computed numerically).

The simplex method

This method, implemented in **QuantLib::Optimization::Simplex**(p. 603), is rather raw and requires quite a lot of computing resources, but it has the advantage that it does not need any evaluation of the cost function's gradient, and that it is quite easily implemented. First, we must choose N+1 starting points, given here by a starting point $\mathbf{P}_0$ and N points such that

$$\mathbf{P_i} = \mathbf{P}_0 + \lambda \mathbf{e_i},$$

where λ is the problem's characteristic length scale). These points will form a geometrical form called simplex. The principle of the downhill simplex method is, at each iteration, to move the worst point (highest cost function value) through the opposite face to a better point. When the simplex seems to be constrained in a valley, it will be contracted downhill, keeping the best point unchanged.

The conjugate gradient method

We'll now continue with a bit more sophisticated method, implemented in **QuantLib::Optimization::ConjugateGradient**(p. 268) . At each step, we minimize (using Armijo's

line search algorithm, implemented in **QuantLib::Optimization::ArmijoLineSearch**(p. 172)) the function along a line defined by

$$\mathbf{d_i} = -\nabla f(\mathbf{x_i}) + \frac{\left\|\nabla f(\mathbf{x_i})\right\|^2}{\left\|\nabla f(\mathbf{x_{i-1}})\right\|^2}\mathbf{d_{i-1}},$$

$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0).$$

As we can see, this optimization method requires the knowledge of the gradient of the cost function. see **QuantLib::Optimization::ConjugateGradient**(p. 268) .

2.10 Design patterns

Documentation for this part of the library is in progress.

2.11 Term structures

Documentation for this part of the library is in progress.

The abstract class **QuantLib::TermStructure**(p. 634) provides the common interface to concrete term structure models. Among others, methods are declared which return instantaneous forward rate, discount factor, and zero rate at a given date. Adapter classes are provided which already implement part of the required methods, thus allowing the programmer to define only the non-redundant part. The PiecewiseConstantForwards class is provided as an example in the **QuantLib::TermStructures**(p. 148) namespace.

2.12 Utilities

Documentation for this part of the library is in progress.

Iterators are meant to build a sequence on the fly from one or more other sequences, without having to allocate place for storing it. A couple of examples: suppose we have a function which calculates the average of a sequence, and that for genericity we have implemented it as a template function which takes the beginning and the end of the sequence, so that its declaration is:

```
template <class Iterator>
typename Iterator::value_type
average(const Iterator& begin, const Iterator& end)
```

This kind of genericity allows one to use the same function to calculate the average of a std::vector, a std::list, a **QuantLib::History**(p. 394), any other container, of a subset of any of the former.

Now let's say we have two sequences of numbers, and we want to calculate the average of their products. One approach could be to store the products in another sequence, and to calculate the average of the latter, as in:

```
// we have sequence1 and sequence2 and assume equal size:
// first we store their product in a vector...
std::vector<double> products;
std::transform(sequence1.begin(),sequence1.end(), // first sequence
               sequence2.begin(),                 // second sequence
               std::back_inserter(products),      // output
               std::multiplies<double>());        // operation to perform
// ...then we calculate the average
double result = average(products.begin(),products.end());
```

The above works, however, it might be not particularly efficient since we have to allocate the product vector, quite possibly just to throw it away when the calculation is done.

QuantLib::Utilities::coupling_iterator(p. 274) allows us to do the same thing without allocating the extra vector: what we do is simply:

```
// we have sequence1 and sequence2 and assume equal size:
double result = average(
    make_coupling_iterator(sequence1.begin(),
                           sequence2.begin(),
                           std::multiplies<double()),
    make_coupling_iterator(sequence1.end(),
                           sequence2.end(),
                           std::multiplies<double()));
```

The call to make_coupling_iterator creates an iterator which is really a reference to the two iterators and the operation we passed. Dereferencing such iterator returns the result of applying such operation to the values pointed to by the two contained iterators. Advancing the coupling iterator advances the two underlying ones. One can see how iterating on such iterator generates the products one by one so that they can be processed by `average()`, but does not need allocating memory for storing the results. The product sequence is generated on the fly.

The other iterators share the same principle but have different functionalities:

- combining_iterator is the same as coupling_iterator, but works on N sequences while the latter works on 2;
- filtering_iterator generates the elements of a given sequence which satisfy a given predicate, i.e., it takes a sequence $[x_0, x_1, \ldots]$ and a predicate p and generates the sequence of those x_i for which $p(x_i)$ returns `true`;

- processing_iterator takes a sequence $[x_0, x_1, \ldots]$ and a function f and generates the sequence $[f(x_0), f(x_1), \ldots]$;
- stepping_iterator takes a sequence $[x_0, x_1, \ldots]$ and a step m and generates the sequence $[x_0, x_m, x_{2m}, \ldots]$

Chapter 3

Examples

More examples of using the QuantLib library can be found in chapter 13 of the reference manual. Should your viewer allow it, they can also be reached via the hyperlinks in the following list.

3.1 QuantLib Examples

Here is a list of all examples:

- **AmericanOption.cpp**
- **BermudanSwaption.cpp**
- **DiscreteHedging.cpp**
- **EuropeanOption.cpp**
- **history_iterators.cpp**
- **swapvaluation.cpp**

Part II

Reference Manual

Chapter 4

QuantLib Module Index

4.1 QuantLib Modules

Here is a list of all modules:

Global QuantLib macros 95
Math functions 97
Numeric limits 98
Time functions 99
String functions 100
Character functions 101
Input/output functions 102
Min and max functions 103
Template capabilities 104
Iterator support 106

Chapter 5

QuantLib Namespace Index

5.1 QuantLib Namespace List

Here is a list of all documented namespaces with brief descriptions:

QuantLib (A.k.a. the QuantLib Foundation) . . . 107
QuantLib::Calendars (Specialized **Calendar**(p. 235) classes) . . . 117
QuantLib::CashFlows (Concrete implementations of the **CashFlow**(p. 252) interface) . . . 119
QuantLib::DayCounters (Specialized **DayCounter**(p. 292) classes) . . . 124
QuantLib::FiniteDifferences (Finite difference framework) . . . 125
QuantLib::Indexes (Concrete implementations of the **Index**(p. 417) interface) . . . 127
QuantLib::Instruments (Concrete implementations of the **Instrument**(p. 422) interface) 128
QuantLib::Math (Mathematical functions and classes) . . . 131
QuantLib::MonteCarlo (Monte Carlo framework) . . . 135
QuantLib::Patterns (Implementations of design patterns) . . . 138
QuantLib::Pricers (Pricing models for options) . . . 139
QuantLib::RandomNumbers (Random Number Generators and Low Discrepancy Sequences) . . . 142
QuantLib::ShortRateModels (Implementations of short-rate models) . . . 144
QuantLib::Solvers1D (Implementations of the **Solver1D**(p. 608) interface) . . . 147
QuantLib::TermStructures (Concrete implementations of the **TermStructure**(p. 634) interface) . . . 148
QuantLib::Utilities (Classes and functions of general utility) . . . 150
QuantLib::VolTermStructures (Concrete implementations of Volatility Term Structures) 151

Chapter 6

QuantLib Hierarchical Index

6.1 QuantLib Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AcyclicVisitor . . . 156
  BPSCalculator . . . 228
AffineModel . . . 158
  G2 . . . 371
  OneFactorAffineModel . . . 528
    CoxIngersollRoss . . . 278
      ExtendedCoxIngersollRoss . . . 336
    Vasicek . . . 669
      HullWhite . . . 400
Arguments . . . 169
  CapFloor::arguments . . . 249
  SimpleSwap::arguments . . . 601
    Swaption::arguments . . . 627
  VanillaOption::arguments . . . 667
    BarrierOption::arguments . . . 185
    BinaryOption::arguments . . . 200
    CliquetOption::arguments . . . 260
Array . . . 173
ArrayFormatter . . . 176
Barrier . . . 180
Binary . . . 195
BlackKarasinski::Dynamics . . . 210
BoundaryCondition . . . 224
BoundaryCondition< TridiagonalOperator > . . . 224
  DirichletBC . . . 300
  NeumannBC . . . 511
BoxMullerGaussianRng . . . 227
Bridge . . . 230
Bridge< Calendar, CalendarImpl > . . . 230
  Calendar . . . 235
    Budapest . . . 233

Frankfurt . . . 367
Helsinki . . . 392
Johannesburg . . . 434
JointCalendar . . . 435
London . . . 467
Milan . . . 499
NewYork . . . 514
NullCalendar . . . 520
Oslo . . . 536
Stockholm . . . 616
Sydney . . . 631
TARGET . . . 633
Tokyo . . . 643
Toronto . . . 645
Warsaw . . . 673
Wellington . . . 674
Zurich . . . 685
Bridge< Constraint, ConstraintImpl > . . . 230
Constraint . . . 270
BoundaryConstraint . . . 226
CompositeConstraint . . . 266
NoConstraint . . . 515
PositiveConstraint . . . 556
Bridge< DayCounter, DayCounterImpl > . . . 230
DayCounter . . . 292
Actual360 . . . 153
Actual365 . . . 154
ActualActual . . . 155
SimpleDayCounter . . . 597
Thirty360 . . . 639
Bridge< Parameter, ParameterImpl > . . . 230
Parameter . . . 539
ConstantParameter . . . 269
NullParameter . . . 521
PiecewiseConstantParameter . . . 552
TermStructureFittingParameter . . . 638
ExtendedCoxIngersollRoss::FittingParameter . . . 339
G2::FittingParameter . . . 373
HullWhite::FittingParameter . . . 403
BrownianBridge . . . 231
CalendarImpl . . . 239
Calendar::WesternImpl . . . 238
CalibrationSet . . . 242
CLGaussianRng . . . 256
CliquetOption . . . 258
CliquetOption . . . 259
combining_iterator . . . 264
ConstraintImpl . . . 271
CostFunction . . . 273
LeastSquareFunction . . . 446
coupling_iterator . . . 274
CoxIngersollRoss::Dynamics . . . 280

ExtendedCoxIngersollRoss::Dynamics . . . 338
CubicSpline . . . 283
CumulativeNormalDistribution . . . 285
CuriouslyRecurringTemplate . . . 286
Solver1D . . . 608
CuriouslyRecurringTemplate< Bisection > . . . 286
Solver1D< Bisection > . . . 608
Bisection . . . 205
CuriouslyRecurringTemplate< Brent > . . . 286
Solver1D< Brent > . . . 608
Brent . . . 229
CuriouslyRecurringTemplate< FalsePosition > . . . 286
Solver1D< FalsePosition > . . . 608
FalsePosition . . . 342
CuriouslyRecurringTemplate< Newton > . . . 286
Solver1D< Newton > . . . 608
Newton . . . 512
CuriouslyRecurringTemplate< NewtonSafe > . . . 286
Solver1D< NewtonSafe > . . . 608
NewtonSafe . . . 513
CuriouslyRecurringTemplate< Ridder > . . . 286
Solver1D< Ridder > . . . 608
Ridder . . . 582
CuriouslyRecurringTemplate< Secant > . . . 286
Solver1D< Secant > . . . 608
Secant . . . 589
CurrencyFormatter . . . 287
Date . . . 288
DateFormatter . . . 291
DayCounterImpl . . . 294
DiffusionProcess . . . 298
BlackScholesProcess . . . 214
OrnsteinUhlenbeckProcess . . . 534
SquareRootProcess . . . 610
DiscretizedAsset . . . 308
DiscretizedDiscountBond . . . 310
DiscretizedOption . . . 311
Disposable . . . 313
DoubleFormatter . . . 315
EndCriteria . . . 321
Error . . . 325
AssertionFailedError . . . 177
IllegalArgumentError . . . 406
IllegalResultError . . . 407
IndexError . . . 421
OutOfMemoryError . . . 537
PostconditionNotSatisfiedError . . . 557
PreconditionNotSatisfiedError . . . 558
ErrorFunction . . . 326
EuroFormatter . . . 328

Exercise . . . 334
AmericanExercise . . . 162
BermudanExercise . . . 189
EuropeanExercise . . . 329
FdBermudanOption . . . 344
FdDividendShoutOption . . . 348
filtering_iterator . . . 351
FiniteDifferenceModel . . . 352
ForwardOptionArguments . . . 359
GammaFunction . . . 374
GaussianStatistics . . . 375
GeneralStatistics . . . 378
GenericRiskStatistics . . . 383
HaltonRsg . . . 389
Handle . . . 390
Handle< BlackModel > . . . 390
Handle< Impl > . . . 390
Handle< Link< TermStructure > > . . . 390
RelinkableHandle< TermStructure > . . . 579
Handle< Link< Type > > . . . 390
RelinkableHandle . . . 579
Handle< MonteCarlo::MonteCarloModel< MonteCarlo::MultiAsset_old< MonteCarlo::PseudoRandomSequence_old >, Math::Statistics > > . . . 390
Handle< MonteCarlo::MonteCarloModel< MonteCarlo::SingleAsset< RNG >, S > > . . . 390
Handle< MonteCarlo::MonteCarloModel< MonteCarlo::SingleAsset_old< MonteCarlo::PseudoRandom_old >, Math::Statistics > > . . . 390
Handle< path_generator_type > . . . 390
Handle< path_pricer_type > . . . 390
Handle< ShortRateModels::OneFactorAffineModel > . . . 390
Handle< TermStructure > . . . 390
History . . . 394
History::const_iterator . . . 397
History::Entry . . . 399
HullWhite::Dynamics . . . 402
ICGaussianRng . . . 404
ICGaussianRsg . . . 405
IncrementalStatistics . . . 414
IntegerFormatter . . . 425
Interpolation . . . 427
CubicSplineInterpolation . . . 284
LinearInterpolation . . . 452
LogLinearInterpolation . . . 466
Interpolation2D . . . 429
BicubicSplineInterpolation . . . 191
BilinearInterpolation . . . 193
InverseCumulativeNormal . . . 431
KnuthUniformRng . . . 437
KronrodIntegral . . . 438
LeastSquareProblem . . . 447
LecuyerUniformRng . . . 448
LexicographicalView . . . 449
Linear . . . 451

LineSearch . . . 453
ArmijoLineSearch . . . 172
LogLinear . . . 465
lowest_category_iterator . . . 468
MakeSchedule . . . 469
Matrix . . . 471
McPricer . . . 490
McBasket . . . 477
McCliquetOption . . . 480
McDiscreteArithmeticAPO . . . 481
McDiscreteArithmeticASO . . . 482
McEuropean . . . 483
McEverest . . . 485
McHimalaya . . . 486
McMaxBasket . . . 487
McPagoda . . . 488
McPerformanceOption . . . 489
McPricer< MonteCarlo::MultiAsset_old< MonteCarlo::PseudoRandomSequence_old > > 490
McPricer< MonteCarlo::SingleAsset_old< MonteCarlo::PseudoRandom_old > > . . . 490
McSimulation . . . 492
MCBarrierEngine . . . 475
MCBinaryEngine . . . 478
MCVanillaEngine . . . 494
MCEuropeanEngine . . . 484
McSimulation< MonteCarlo::SingleAsset< RNG >, S > . . . 492
MersenneTwisterUniformRng . . . 496
Method . . . 497
ConjugateGradient . . . 268
Simplex . . . 603
SteepestDescent . . . 611
MixedScheme . . . 500
CrankNicolson . . . 282
ExplicitEuler . . . 335
ImplicitEuler . . . 408
MonteCarloModel . . . 504
MonteCarloModel< MonteCarlo::MultiAsset_old< MonteCarlo::PseudoRandom-Sequence_old >, Math::Statistics > . . . 504
MonteCarloModel< MonteCarlo::SingleAsset< RNG >, S > . . . 504
MonteCarloModel< MonteCarlo::SingleAsset_old< MonteCarlo::PseudoRandom_old >, Math::Statistics > . . . 504
MoroInverseCumulativeNormal . . . 505
MultiPath . . . 506
MultiPathGenerator . . . 507
MultiPathGenerator_old . . . 508
MultivariateAccumulator . . . 509
NonLinearLeastSquare . . . 516
NormalDistribution . . . 518
Null . . . 519
NumericalMethod . . . 522
Lattice . . . 439
BlackScholesLattice . . . 213
Lattice2D . . . 441

TwoFactorModel::ShortRateTree . . . 659
OneFactorModel::ShortRateTree . . . 531
Observable . . . 523
BlackModel . . . 211
BlackVolTermStructure . . . 222
BlackVarianceTermStructure . . . 220
BlackVarianceCurve . . . 216
BlackVarianceSurface . . . 218
ImpliedVolTermStructure . . . 411
BlackVolatilityTermStructure . . . 221
BlackConstantVol . . . 207
CapFlatVolatilityStructure . . . 244
CapFlatVolatilityVector . . . 246
CapletForwardVolatilityStructure . . . 251
CashFlow . . . 252
Coupon . . . 276
FixedRateCoupon . . . 353
FloatingRateCoupon . . . 355
IndexedCoupon . . . 419
InArrearIndexedCoupon . . . 413
UpFrontIndexedCoupon . . . 660
ParCoupon . . . 541
ShortFloatingRateCoupon . . . 594
SimpleCashFlow . . . 596
Index . . . 417
Xibor . . . 675
AUDLibor . . . 179
CADLibor . . . 234
CHFLibor . . . 255
Euribor . . . 327
GBPLibor . . . 377
JPYLibor . . . 436
USDLibor . . . 661
ZARLibor . . . 678
Link . . . 455
Link< TermStructure > . . . 455
LocalVolTermStructure . . . 463
LocalConstantVol . . . 457
LocalVolCurve . . . 459
LocalVolSurface . . . 461
MarketElement . . . 470
CompositeMarketElement . . . 267
DerivedMarketElement . . . 297
SimpleMarketElement . . . 598
LazyObject . . . 443
Instrument . . . 422
CapFloor . . . 247
Cap . . . 243
Collar . . . 263
Floor . . . 357
Stock . . . 615
Swap . . . 621

SimpleSwap . . . 599
Option . . . 533
BarrierOption . . . 182
BinaryOption . . . 197
Swaption . . . 625
VanillaOption . . . 664
ForwardVanillaOption . . . 365
QuantoVanillaOption . . . 572
QuantoForwardVanillaOption . . . 566
PiecewiseFlatForward . . . 553
GenericModelEngine . . . 382
AnalyticalCapFloor . . . 164
BlackCapFloor . . . 206
BlackSwaption . . . 215
JamshidianSwaption . . . 432
LatticeShortRateModelEngine . . . 442
TreeCapFloor . . . 649
TreeSwaption . . . 650
LatticeShortRateModelEngine< Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 442
LatticeShortRateModelEngine< Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 442
GenericModelEngine< BlackModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< BlackModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
GenericModelEngine< ShortRateModels::AffineModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Arguments, Results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
GenericModelEngine< ShortRateModels::OneFactorAffineModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
CalibrationHelper . . . 240
Model . . . 502
OneFactorModel . . . 529
BlackKarasinski . . . 209
OneFactorAffineModel . . . 528
TwoFactorModel . . . 657
G2 . . . 371
SwaptionVolatilityStructure . . . 630
SwaptionVolatilityMatrix . . . 629
TermStructure . . . 634
DiscountStructure . . . 303
AffineTermStructure . . . 159
DiscountCurve . . . 301
ExtendedDiscountCurve . . . 340
ImpliedTermStructure . . . 409
ForwardRateStructure . . . 361
ForwardSpreadedTermStructure . . . 363

PiecewiseFlatForward . . . 553
ZeroYieldStructure . . . 683
DriftTermStructure . . . 318
QuantoTermStructure . . . 570
ZeroCurve . . . 679
ZeroSpreadedTermStructure . . . 681
RateHelper . . . 577
DepositRateHelper . . . 295
FraRateHelper . . . 368
FuturesRateHelper . . . 370
SwapRateHelper . . . 623
Observer . . . 525
BlackModel . . . 211
IndexedCoupon . . . 419
ParCoupon . . . 541
CompositeMarketElement . . . 267
DerivedMarketElement . . . 297
Xibor . . . 675
Link . . . 455
Link< TermStructure > . . . 455
LazyObject . . . 443
GenericModelEngine . . . 382
GenericModelEngine< BlackModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< BlackModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
GenericModelEngine< ShortRateModels::AffineModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Arguments, Results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
GenericModelEngine< ShortRateModels::OneFactorAffineModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
CalibrationHelper . . . 240
Model . . . 502
AffineTermStructure . . . 159
DriftTermStructure . . . 318
ExtendedDiscountCurve . . . 340
ForwardSpreadedTermStructure . . . 363
ImpliedTermStructure . . . 409
QuantoTermStructure . . . 570
RateHelper . . . 577
ZeroSpreadedTermStructure . . . 681
BlackConstantVol . . . 207
BlackVarianceCurve . . . 216
BlackVarianceSurface . . . 218
ImpliedVolTermStructure . . . 411
LocalConstantVol . . . 457
LocalVolCurve . . . 459
LocalVolSurface . . . 461
OneFactorModel::ShortRateDynamics . . . 530

ParameterImpl . . . 540
Path . . . 543
PathGenerator . . . 544
PathGenerator_old . . . 545
PathPricer . . . 546
PathPricer< Path > . . . 546
BarrierPathPricer . . . 187
BiasedBarrierPathPricer . . . 190
BinaryPathPricer . . . 202
CliquetOptionPathPricer . . . 261
EuropeanPathPricer . . . 331
PathPricer_old . . . 547
PathPricer_old< MultiPath > . . . 547
BasketPathPricer_old . . . 188
EverestPathPricer_old . . . 333
HimalayaPathPricer_old . . . 393
MaxBasketPathPricer_old . . . 474
PagodaPathPricer_old . . . 538
PathPricer_old< Path > . . . 547
ArithmeticAPOPathPricer_old . . . 170
ArithmeticASOPathPricer_old . . . 171
CliquetOptionPathPricer_old . . . 262
EuropeanPathPricer_old . . . 332
GeometricAPOPathPricer_old . . . 386
GeometricASOPathPricer_old . . . 387
PerformanceOptionPathPricer_old . . . 550
Payoff . . . 548
StrikedTypePayoff . . . 617
AssetOrNothingPayoff . . . 178
CashOrNothingPayoff . . . 254
PlainVanillaPayoff . . . 555
SupersharePayoff . . . 619
PerformanceOption . . . 549
Period . . . 551
PricingEngine . . . 559
GenericEngine . . . 381
BarrierEngine . . . 181
AnalyticBarrierEngine . . . 166
MCBarrierEngine . . . 475
BinaryEngine . . . 196
AnalyticAmericanBinaryEngine . . . 165
AnalyticEuropeanBinaryEngine . . . 167
MCBinaryEngine . . . 478
CliquetEngine . . . 257
ForwardEngine . . . 358
ForwardPerformanceEngine . . . 360
GenericModelEngine . . . 382
GenericModelEngine< BlackModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< BlackModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382

GenericModelEngine< ShortRateModels::AffineModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::CapFloor::arguments, Instruments::CapFloor::results > . . . 382
GenericModelEngine< ShortRateModels::Model, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
GenericModelEngine< ShortRateModels::OneFactorAffineModel, Instruments::Swaption::arguments, Instruments::Swaption::results > . . . 382
QuantoEngine . . . 565
VanillaEngine . . . 663
AmericanMCVanillaEngine . . . 163
AnalyticEuropeanEngine . . . 168
BinomialVanillaEngine . . . 204
IntegralEngine . . . 426
MCVanillaEngine . . . 494
GenericEngine< Arguments, Results > . . . 381
GenericModelEngine< ShortRateModels::Model, Arguments, Results > . . . 382
GenericEngine< Instruments::BarrierOption::arguments, Instruments::BarrierOption::results > . . . 381
GenericEngine< Instruments::BinaryOption::arguments, Instruments::BinaryOption::results > . . . 381
GenericEngine< Instruments::CapFloor::arguments, Instruments::CapFloor::results > 381
GenericEngine< Instruments::CliquetOption::arguments, Instruments::VanillaOption::results > . . . 381
GenericEngine< Instruments::ForwardOptionArguments< ArgumentsType >, ResultsType > . . . 381
GenericEngine< Instruments::QuantoOptionArguments< ArgumentsType >, Instruments::QuantoOptionResults< ResultsType > > . . . 381
GenericEngine< Instruments::Swaption::arguments, Instruments::Swaption::results > 381
GenericEngine< Instruments::VanillaOption::arguments, Instruments::VanillaOption::results > . . . 381
PrimeNumbers . . . 560
Problem . . . 561
processing_iterator . . . 563
QuantoOptionArguments . . . 568
QuantoOptionResults . . . 569
RandomArrayGenerator . . . 574
RandomSequenceGenerator . . . 575
RateFormatter . . . 576
Results . . . 581
Greeks . . . 388
VanillaOption::results . . . 668
BarrierOption::results . . . 186
BinaryOption::results . . . 201
Value . . . 662
CapFloor::results . . . 250
SimpleSwap::results . . . 602
Swaption::results . . . 628
VanillaOption::results . . . 668
RiskMeasures . . . 583
RiskStatistics_old . . . 585
Sample . . . 587
Schedule . . . 588

SegmentIntegral . . . 590
SequenceStatistics . . . 591
SequenceStatistics< Statistics > . . . 591
DiscrepancyStatistics . . . 305
Short . . . 593
SingleAssetOption . . . 605
BarrierOption . . . 184
BinaryOption . . . 199
DiscreteGeometricAPO . . . 306
DiscreteGeometricASO . . . 307
EuropeanOption . . . 330
ContinuousGeometricAPO . . . 272
FdDividendEuropeanOption . . . 347
FdBsmOption . . . 345
FdEuropean . . . 349
FdStepConditionOption . . . 350
FdAmericanOption . . . 343
SobolRsg . . . 607
StepCondition . . . 612
AmericanCondition . . . 161
ShoutCondition . . . 595
StepCondition< Array > . . . 612
stepping_iterator . . . 613
StringFormatter . . . 618
SVD . . . 620
SymmetricSchurDecomposition . . . 632
TermStructureConsistentModel . . . 637
BlackKarasinski . . . 209
ExtendedCoxIngersollRoss . . . 336
G2 . . . 371
HullWhite . . . 400
TimeBasket . . . 641
TimeGrid . . . 642
TrapezoidIntegral . . . 646
SimpsonIntegral . . . 604
Tree . . . 648
BinomialTree . . . 203
EqualJumpsBinomialTree . . . 323
CoxRossRubinstein . . . 281
Trigeorgis . . . 654
EqualProbabilitiesBinomialTree . . . 324
AdditiveEQPBinomialTree . . . 157
JarrowRudd . . . 433
Tian . . . 640
TrinomialTree . . . 656
TridiagonalOperator . . . 651
BSMOperator . . . 232
DMinus . . . 314
DPlus . . . 316
DPlusDMinus . . . 317
DZero . . . 320

OneFactorOperator 532
TridiagonalOperator::TimeSetter 653
TrinomialBranching 655
TwoFactorModel::ShortRateDynamics 658
Vasicek::Dynamics 671
Visitor 672
Visitor< BlackVolTermStructure > 672
Visitor< CashFlow > 672
BPSCalculator 228
Visitor< Coupon > 672
BPSCalculator 228
Visitor< FixedRateCoupon > 672
Visitor< VolTermStructures::BlackConstantVol > 672
Visitor< VolTermStructures::BlackVarianceCurve > 672
XiborManager 677

Chapter 7

QuantLib Class Index

7.1 QuantLib Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Actual360 (Actual/360 day count convention) . . . 153
Actual365 (Actual/365 day count convention) . . . 154
ActualActual (Actual/Actual day count) . . . 155
AcyclicVisitor (Degenerate base class for the Acyclic **Visitor**(p. 672) pattern) . . . 156
AdditiveEQPBinomialTree (Additive equal probabilities binomial tree) . . . 157
AffineModel (Affine model class) . . . 158
AffineTermStructure (Term-structure implied by an affine model) . . . 159
AmericanCondition . . . 161
AmericanExercise (American exercise class) . . . 162
AmericanMCVanillaEngine (Least-square Monte Carlo engine) . . . 163
AnalyticalCapFloor (Analytical pricer for cap/floor) . . . 164
AnalyticAmericanBinaryEngine (Pricing engine for American binary options using analytic formulae) . . . 165
AnalyticBarrierEngine (Pricing engine for **Barrier**(p. 180) options using analytical formulae) . . . 166
AnalyticEuropeanBinaryEngine (Pricing engine for European binary options using analytic formulae) . . . 167
AnalyticEuropeanEngine (Pricing engine for European options using analytical formulae) . . . 168
Arguments (Base class for generic argument groups) . . . 169
ArithmeticAPOPathPricer_old (path pricer for arithmetic average price option) . . . 170
ArithmeticASOPathPricer_old (path pricer for arithmetic average strike Asian options) 171
ArmijoLineSearch . . . 172
Array (1-D array used in linear algebra) . . . 173
ArrayFormatter (Formats arrays for output) . . . 176
AssertionFailedError (Specialized error) . . . 177
AssetOrNothingPayoff (**Binary**(p. 195) Asset-Or-Nothing option payoff) . . . 178
AUDLibor (AUD Libor index (Also known as SIBOR, check settlement days)) . . . 179
Barrier (Placeholder for enumerated barrier types) . . . 180
BarrierEngine (**Barrier**(p. 180) engine base class) . . . 181
BarrierOption (**Barrier**(p. 180) option on a single asset) . . . 182
BarrierOption (**Barrier**(p. 180) option) . . . 184
BarrierOption::arguments (**Arguments**(p. 169) for barrier option calculation) . . . 185

BarrierOption::results (results from barrier option calculation) . . . 186
BarrierPathPricer (path pricer for **Barrier**(p. 180) options) . . . 187
BasketPathPricer_old (Multipath pricer for European-type basket option) . . . 188
BermudanExercise (Bermudan exercise class) . . . 189
BiasedBarrierPathPricer (Biased, but simple, path pricer for **Barrier**(p. 180) options) . . . 190
BicubicSplineInterpolation (Bicubic spline interpolation between discrete points) . . . 191
BilinearInterpolation (Bilinear interpolation between discrete points) . . . 193
Binary (Placeholder for enumerated binary types) . . . 195
BinaryEngine (**Binary**(p. 195) engine base class) . . . 196
BinaryOption (**Binary**(p. 195) option on a single asset) . . . 197
BinaryOption (**Binary**(p. 195) (digital) option) . . . 199
BinaryOption::arguments (arguments for binary option calculation) . . . 200
BinaryOption::results (results from binary option calculation) . . . 201
BinaryPathPricer (path pricer for **Binary**(p. 195) options) . . . 202
BinomialTree (Binomial tree base class) . . . 203
BinomialVanillaEngine (Pricing engine for Vanilla options using binomial trees) . . . 204
Bisection (bisection 1-D solver) . . . 205
BlackCapFloor (CapFloor priced by the Black formula) . . . 206
BlackConstantVol (Constant Black volatility, no time-strike dependence) . . . 207
BlackKarasinski (Standard Black-Karasinski model class) . . . 209
BlackKarasinski::Dynamics (Short-rate dynamics in the Black-Karasinski model) . . . 210
BlackModel (Black-model for vanilla interest-rate derivatives) . . . 211
BlackScholesLattice (Simple binomial lattice approximating the Black-Scholes model) . 213
BlackScholesProcess (Black-Scholes diffusion process class) . . . 214
BlackSwaption (Swaption priced by the Black formula) . . . 215
BlackVarianceCurve (Black volatility curve modelled as variance curve) . . . 216
BlackVarianceSurface (Black volatility surface modelled as variance surface) . . . 218
BlackVarianceTermStructure (Black Variance term structure) . . . 220
BlackVolatilityTermStructure (Black Volatility term structure) . . . 221
BlackVolTermStructure (Black Volatility Term structure) . . . 222
BoundaryCondition (Abstract boundary condition class for finite difference problems) 224
BoundaryConstraint (**Constraint**(p. 270) imposing all arguments to be in [low,high]) . 226
BoxMullerGaussianRng (Gaussian random number generator) . . . 227
BPSCalculator (Basis point sensitivity (BPS) calculator) . . . 228
Brent (Brent 1-D solver) . . . 229
Bridge (The **Bridge**(p. 230) pattern made explicit) . . . 230
BrownianBridge (Builds Wiener process paths using Gaussian variates) . . . 231
BSMOperator (Black-Scholes-Merton differential operator) . . . 232
Budapest (Budapest calendar) . . . 233
CADLibor (CAD Libor index (Also known as CDOR)) . . . 234
Calendar (calendar class) . . . 235
Calendar::WesternImpl . . . 238
CalendarImpl (Abstract base class for calendar implementations) . . . 239
CalibrationHelper (Liquid market instrument used during calibration) . . . 240
CalibrationSet (Set of calibration instruments) . . . 242
Cap (Concrete cap class) . . . 243
CapFlatVolatilityStructure (Cap/floor flat volatility structure) . . . 244
CapFlatVolatilityVector (Cap/floor at-the-money flat volatility vector) . . . 246
CapFloor (Base class for cap-like instruments) . . . 247
CapFloor::arguments (**Arguments**(p. 169) for cap/floor calculation) . . . 249
CapFloor::results (results from cap/floor calculation) . . . 250
CapletForwardVolatilityStructure (Caplet/floorlet forward volatility structure) . . . 251
CashFlow (Base class for cash flows) . . . 252
CashOrNothingPayoff (**Binary**(p. 195) Cash-Or-Nothing option payoff) . . . 254

CHFLibor (CHF Libor index (Also known as ZIBOR)) . . . 255
CLGaussianRng (Gaussian random number generator) . . . 256
CliquetEngine (Cliquet engine base class) . . . 257
CliquetOption (Currently just a container for arguments and results) . . . 258
CliquetOption (Cliquet (Ratchet) option) . . . 259
CliquetOption::arguments (**Arguments**(p. 169) for cliquet option calculation) . . . 260
CliquetOptionPathPricer (path pricer for cliquet options) . . . 261
CliquetOptionPathPricer_old (path pricer for cliquet options) . . . 262
Collar (Concrete cap class) . . . 263
combining_iterator (Iterator mapping a function to a set of underlying sequences) . . . 264
CompositeConstraint (**Constraint**(p. 270) enforcing both given sub-constraints) . . . 266
CompositeMarketElement (Market element whose value depends on two other market element) . . . 267
ConjugateGradient (Multi-dimensional Conjugate Gradient class) . . . 268
ConstantParameter (Standard constant parameter $a(t) = a$) . . . 269
Constraint (Base constraint class) . . . 270
ConstraintImpl (Base class for **Constraint**(p. 270) implementations) . . . 271
ContinuousGeometricAPO (Continuous Geometric Average Price **Option**(p. 533) (European exercise)) . . . 272
CostFunction (Cost function abstract class for optimization problem) . . . 273
coupling_iterator (Iterator mapping a function to a pair of underlying sequences) . . . 274
Coupon (coupon accruing over a fixed period) . . . 276
CoxIngersollRoss (Cox-Ingersoll-Ross model class) . . . 278
CoxIngersollRoss::Dynamics (**Dynamics**(p. 280) of the short-rate under the Cox-Ingersoll-Ross model) . . . 280
CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree) 281
CrankNicolson (Crank-Nicolson scheme for finite difference methods) . . . 282
CubicSpline (Cubic-spline interpolation traits) . . . 283
CubicSplineInterpolation (Cubic spline interpolation between discrete points) . . . 284
CumulativeNormalDistribution (Cumulative normal distribution function) . . . 285
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern) . 286
CurrencyFormatter (Formats currencies for output) . . . 287
Date (Concrete date class) . . . 288
DateFormatter (Formats dates for output) . . . 291
DayCounter (Day counter class) . . . 292
DayCounterImpl (Abstract base class for day counter implementations) . . . 294
DepositRateHelper (Deposit rate) . . . 295
DerivedMarketElement (Market element whose value depends on another market element) . . . 297
DiffusionProcess (Diffusion process class) . . . 298
DirichletBC (Neumann boundary condition (i.e., constant value)) . . . 300
DiscountCurve (Term structure based on loglinear interpolation of discount factors) . . 301
DiscountStructure (Discount factor term structure) . . . 303
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation) . . . 305
DiscreteGeometricAPO (Discrete Geometric Average Price Asian **Option**(p. 533) (European style)) . . . 306
DiscreteGeometricASO (Discrete Geometric Average Strike Asian **Option**(p. 533) (European style)) . . . 307
DiscretizedAsset (Discretized asset class used by numerical methods) . . . 308
DiscretizedDiscountBond (Useful discretized discount bond asset) . . . 310
DiscretizedOption (Discretized option on another asset) . . . 311
Disposable (Generic disposable object with move semantics) . . . 313
DMinus (D_- matricial representation) . . . 314
DoubleFormatter (Formats doubles for output) . . . 315

DPlus (D_+ matricial representation) . . . 316
DPlusDMinus (D_+D_- matricial representation) . . . 317
DriftTermStructure (Drift term structure) . . . 318
DZero (D_0 matricial representation) . . . 320
EndCriteria . . . 321
EqualJumpsBinomialTree (Base class for equal jumps binomial tree) . . . 323
EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree) . . . 324
Error (Base error class) . . . 325
ErrorFunction (**Error**(p. 325) function) . . . 326
Euribor (Euribor index) . . . 327
EuroFormatter (Formats amounts in Euro for output) . . . 328
EuropeanExercise (European exercise class) . . . 329
EuropeanOption (Black-Scholes-Merton European option) . . . 330
EuropeanPathPricer (path pricer for European options) . . . 331
EuropeanPathPricer_old (path pricer for European options) . . . 332
EverestPathPricer_old (path pricer for European-type Everest option) . . . 333
Exercise (**Exercise**(p. 334) class (American, Bermudan or European)) . . . 334
ExplicitEuler (Forward Euler scheme for finite difference methods) . . . 335
ExtendedCoxIngersollRoss (Extended Cox-Ingersoll-Ross model class) . . . 336
ExtendedCoxIngersollRoss::Dynamics (Short-rate dynamics in the extended Cox-Ingersoll-Ross model) . . . 338
ExtendedCoxIngersollRoss::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$) . . . 339
ExtendedDiscountCurve (Term structure based on loglinear interpolation of discount factors) . . . 340
FalsePosition (False position 1-D solver) . . . 342
FdAmericanOption . . . 343
FdBermudanOption (Bermudan option) . . . 344
FdBsmOption (Black-Scholes-Merton option priced numerically) . . . 345
FdDividendEuropeanOption (European option with dividends) . . . 347
FdDividendShoutOption (Shout option with dividends) . . . 348
FdEuropean (Example of European option calculated using finite differences) . . . 349
FdStepConditionOption (option executing additional code at each time step) . . . 350
filtering_iterator (Iterator filtering undesired data) . . . 351
FiniteDifferenceModel (Generic finite difference model) . . . 352
FixedRateCoupon (coupon paying a fixed interest rate) . . . 353
FloatingRateCoupon (coupon at par on a term structure) . . . 355
Floor (Concrete floor class) . . . 357
ForwardEngine (Forward engine base class) . . . 358
ForwardOptionArguments (**Arguments**(p. 169) for forward (strike-resetting) option calculation) . . . 359
ForwardPerformanceEngine (Forward Performance engine base class) . . . 360
ForwardRateStructure (Forward rate term structure) . . . 361
ForwardSpreadedTermStructure (Term structure with added spread on the instantaneous forward rate) . . . 363
ForwardVanillaOption (Forward version of a vanilla option) . . . 365
Frankfurt (Frankfurt calendar) . . . 367
FraRateHelper (Forward rate agreement) . . . 368
FuturesRateHelper (Interest Rate Futures) . . . 370
G2 (Two-additive-factor gaussian model class) . . . 371
G2::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$) . . . 373
GammaFunction (Gamma function class) . . . 374
GaussianStatistics (Statistics tool for gaussian-assumption risk measures) . . . 375
GBPLibor (GBP Libor index) . . . 377

GeneralStatistics (Statistics tool) . . . 378
GenericEngine (Template base class for option pricing engines) . . . 381
GenericModelEngine (Base class for some pricing engine on a particular model) . . . 382
GenericRiskStatistics (Empirical-distribution risk measures) . . . 383
GeometricAPOPathPricer_old (path pricer for geometric average price option) . . . 386
GeometricASOPathPricer_old (path pricer for geometric average strike option) . . . 387
Greeks (Additional option results) . . . 388
HaltonRsg (Halton low-discrepancy sequence generator) . . . 389
Handle (Reference-counted pointer) . . . 390
Helsinki (Helsinki calendar) . . . 392
HimalayaPathPricer_old (Multipath pricer for European-type Himalaya option) . . . 393
History (Container for historical data) . . . 394
History::const_iterator (Random access iterator on history entries) . . . 397
History::Entry (Single datum in history) . . . 399
HullWhite (Single-factor Hull-White (extended **Vasicek**(p. 669)) model class) . . . 400
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model) . . . 402
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$) . . . 403
ICGaussianRng (Inverse cumulative Gaussian random number generator) . . . 404
ICGaussianRsg (Inverse cumulative Gaussian random sequence generator) . . . 405
IllegalArgumentError (Specialized error) . . . 406
IllegalResultError (Specialized error) . . . 407
ImplicitEuler (Backward Euler scheme for finite difference methods) . . . 408
ImpliedTermStructure (Implied term structure at a given date in the future) . . . 409
ImpliedVolTermStructure (Implied vol term structure at a given date in the future) . . . 411
InArrearIndexedCoupon (in arrear indexed coupon class) . . . 413
IncrementalStatistics (Statistics tool based on incremental accumulation) . . . 414
Index (Purely virtual base class for indexes) . . . 417
IndexedCoupon (base indexed coupon class) . . . 419
IndexError (Specialized error) . . . 421
Instrument (Abstract instrument class) . . . 422
IntegerFormatter (Formats integers for output) . . . 425
IntegralEngine (Pricing engine for Vanilla options using integral approach) . . . 426
Interpolation (Abstract base class for 1-D interpolations) . . . 427
Interpolation2D (Abstract base class for 2-D interpolations) . . . 429
InverseCumulativeNormal (Inverse cumulative normal distribution function) . . . 431
JamshidianSwaption (Jamshidian swaption pricer) . . . 432
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree) . . . 433
Johannesburg (Johannesburg calendar) . . . 434
JointCalendar (Joint calendar) . . . 435
JPYLibor (JPY Libor index (Also known as TIBOR, check settlement days)) . . . 436
KnuthUniformRng (Uniform random number generator) . . . 437
KronrodIntegral (Integral of a 1-dimensional function using the Gauss-Kronrod method) . . . 438
Lattice (Lattice-method base class) . . . 439
Lattice2D (Two-dimensional lattice) . . . 441
LatticeShortRateModelEngine (Engine for a short-rate model specialized on a lattice) . . . 442
LazyObject (Framework for calculation on demand and result caching) . . . 443
LeastSquareFunction . . . 446
LeastSquareProblem (Base class for least square problem) . . . 447
LecuyerUniformRng (Uniform random number generator) . . . 448
LexicographicalView (Lexicographical 2-D view of a contiguous set of data) . . . 449
Linear (**Linear**(p. 451) interpolation traits) . . . 451
LinearInterpolation (**Linear**(p. 451) interpolation between discrete points) . . . 452
LineSearch (Base class for line search) . . . 453
Link (Relinkable access to a **Handle**(p. 390)) . . . 455

LocalConstantVol (Constant local volatility, no time-strike dependence) . . . 457
LocalVolCurve (Local volatility curve derived from a Black curve) . . . 459
LocalVolSurface (Local volatility surface derived from a Black vol surface) . . . 461
LocalVolTermStructure (Local Volatility Term structure) . . . 463
LogLinear (Log-linear interpolation traits) . . . 465
LogLinearInterpolation (Log linear interpolation between discrete points) . . . 466
London (London calendar) . . . 467
lowest_category_iterator (Most generic of two given iterator categories) . . . 468
MakeSchedule (Helper class) . . . 469
MarketElement (Purely virtual base class for market observables) . . . 470
Matrix (matrix used in linear algebra) . . . 471
MaxBasketPathPricer_old (Multipath pricer for European-type basket option) . . . 474
MCBarrierEngine (Pricing engine for **Barrier**(p. 180) options using Monte Carlo) . . . 475
McBasket (Simple example of multi-factor Monte Carlo pricer) . . . 477
MCBinaryEngine (Pricing engine for **Binary**(p. 195) options using Monte Carlo) . . . 478
McCliquetOption (Simple example of Monte Carlo pricer) . . . 480
McDiscreteArithmeticAPO (Example of Monte Carlo pricer using a control variate) . . . 481
McDiscreteArithmeticASO (Example of Monte Carlo pricer using a control variate) . . . 482
McEuropean (Simple example of Monte Carlo pricer) . . . 483
MCEuropeanEngine (European option pricing engine using Monte Carlo simulation) . . . 484
McEverest (Everest-type option pricer) . . . 485
McHimalaya (Himalayan-type option pricer) . . . 486
McMaxBasket (Simple example of multi-factor Monte Carlo pricer) . . . 487
McPagoda (Roofed Asian option) . . . 488
McPerformanceOption (Performance option computed using Monte Carlo simulation) 489
McPricer (Base class for Monte Carlo pricers) . . . 490
McSimulation (Base class for Monte Carlo engines) . . . 492
MCVanillaEngine (Base class for Monte Carlo vanilla option engines) . . . 494
MersenneTwisterUniformRng (Uniform random number generator) . . . 496
Method (Optimization **Method**(p. 497) abstract class for constrained optimization pb) . . . 497
Milan (Milan calendar) . . . 499
MixedScheme (Mixed (explicit/implicit) scheme for finite difference methods) . . . 500
Model (Abstract short-rate model class) . . . 502
MonteCarloModel (General purpose Monte Carlo model for path samples) . . . 504
MoroInverseCumulativeNormal (Moro Inverse cumulative normal distribution class) 505
MultiPath (Single random walk) . . . 506
MultiPathGenerator (Generates a multipath from a random number generator) . . . 507
MultiPathGenerator_old (Generates a multipath from a random number generator) . . . 508
MultivariateAccumulator (A sample accumulator for multivariate analysis) . . . 509
NeumannBC (Neumann boundary condition (i.e., constant derivative)) . . . 511
Newton (Newton 1-D solver) . . . 512
NewtonSafe (Safe Newton 1-D solver) . . . 513
NewYork (New York calendar) . . . 514
NoConstraint (No constraint) . . . 515
NonLinearLeastSquare . . . 516
NormalDistribution (Normal distribution function) . . . 518
Null (Template class providing a null value for a given type) . . . 519
NullCalendar (**Calendar**(p. 235) for reproducing theoretical calculations) . . . 520
NullParameter (**Parameter**(p. 539) which is always zero $a(t) = 0$) . . . 521
NumericalMethod (Numerical method (Tree, Finite Differences) base class) . . . 522
Observable (Object that notifies its changes to a set of observables) . . . 523
Observer (Object that gets notified when a given observable changes) . . . 525
OneFactorAffineModel (Single-factor affine base class) . . . 528
OneFactorModel (Single-factor short-rate model abstract class) . . . 529

OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics) 530
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable) 531
OneFactorOperator (Interest-rate single factor model differential operator) 532
Option (Base option class) 533
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class) 534
Oslo (Oslo calendar) 536
OutOfMemoryError (Specialized error) 537
PagodaPathPricer_old (multipath pricer for pagoda options) 538
Parameter (Base class for model arguments) 539
ParameterImpl (Base class for model parameter implementation) 540
ParCoupon (coupon at par on a term structure) 541
Path (Single factor random walk) 543
PathGenerator (Generates random paths using a sequence generator) 544
PathGenerator_old (Generates random paths from a random number generator) 545
PathPricer (Base class for path pricers) 546
PathPricer_old (Base class for path pricers) 547
Payoff (**Option**(p. 533) payoff base class) 548
PerformanceOption (Performance option) 549
PerformanceOptionPathPricer_old (path pricer for performance options) 550
Period (Time period described by a number of a given time unit) 551
PiecewiseConstantParameter (Piecewise constant parameter) 552
PiecewiseFlatForward (Piecewise flat forward term structure) 553
PlainVanillaPayoff 555
PositiveConstraint (**Constraint**(p. 270) imposing positivity to all arguments) 556
PostconditionNotSatisfiedError (Specialized error) 557
PreconditionNotSatisfiedError (Specialized error) 558
PricingEngine (Base class for pricing engines) 559
PrimeNumbers (Prime numbers calculator) 560
Problem (Constrained optimization problem) 561
processing_iterator (Iterator mapping a unary function to an underlying sequence) 563
QuantoEngine (Quanto engine base class) 565
QuantoForwardVanillaOption (Quanto version of a forward vanilla option) 566
QuantoOptionArguments (**Arguments**(p. 169) for quanto option calculation) 568
QuantoOptionResults (results from quanto option calculation) 569
QuantoTermStructure (Quanto term structure) 570
QuantoVanillaOption (Quanto version of a vanilla option) 572
RandomArrayGenerator (Generates random arrays using a random number generator) 574
RandomSequenceGenerator (Random sequence generator based on a pseudo-random number generator) 575
RateFormatter (Formats rates for output) 576
RateHelper (Base class for rate helpers) 577
RelinkableHandle (Globally accessible relinkable pointer) 579
Results (Base class for generic result groups) 581
Ridder (Ridder 1-D solver) 582
RiskMeasures 583
RiskStatistics_old (Risk analysis tool) 585
Sample (Weighted sample) 587
Schedule (Payment schedule) 588
Secant (secant 1-D solver) 589
SegmentIntegral (Integral of a one-dimensional function) 590
SequenceStatistics (Statistics analysis of N-dimensional (sequence) data) 591
Short (**Short**(p. 593) indexed coupon) 593
ShortFloatingRateCoupon (**Short**(p. 593) coupon at par on a term structure) 594

ShoutCondition (Shout option condition) . . . 595
SimpleCashFlow (Predetermined cash flow) . . . 596
SimpleDayCounter (Simple day counter for reproducing theoretical calculations) . . . 597
SimpleMarketElement (Market element returning a stored value) . . . 598
SimpleSwap (Simple fixed-rate vs Libor swap) . . . 599
SimpleSwap::arguments (**Arguments**(p. 169) for simple swap calculation) . . . 601
SimpleSwap::results (results from swaption calculation) . . . 602
Simplex (Multi-dimensional **Simplex**(p. 603) class) . . . 603
SimpsonIntegral (Integral of a one-dimensional function) . . . 604
SingleAssetOption (Black-Scholes-Merton option) . . . 605
SobolRsg (Sobol low-discrepancy sequence generator) . . . 607
Solver1D (Base class for 1-D solvers) . . . 608
SquareRootProcess (Square-root process class) . . . 610
SteepestDescent (Multi-dimensional steepest-descent class) . . . 611
StepCondition (Condition to be applied at every time step) . . . 612
stepping_iterator (Iterator advancing in constant steps) . . . 613
Stock (Simple stock class) . . . 615
Stockholm (Stockholm calendar) . . . 616
StrikedTypePayoff . . . 617
StringFormatter (Formats strings as lower- or uppercase) . . . 618
SupersharePayoff (**Binary**(p. 195) supershare option payoff) . . . 619
SVD (Singular **Value**(p. 662) Decomposition) . . . 620
Swap (Interest rate swap) . . . 621
SwapRateHelper (Swap rate) . . . 623
Swaption (**Swaption**(p. 625) class) . . . 625
Swaption::arguments (**Arguments**(p. 169) for swaption calculation) . . . 627
Swaption::results (results from swaption calculation) . . . 628
SwaptionVolatilityMatrix (Swaption at-the-money volatility matrix) . . . 629
SwaptionVolatilityStructure (Swaption volatility structure) . . . 630
Sydney (Sydney, calendar (New South Wales, Australia)) . . . 631
SymmetricSchurDecomposition (Symmetric threshold Jacobi algorithm) . . . 632
TARGET (TARGET calendar) . . . 633
TermStructure (Term structure) . . . 634
TermStructureConsistentModel (Term-structure consistent model class) . . . 637
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting) . . . 638
Thirty360 (30/360 day count convention) . . . 639
Tian (**Tian**(p. 640) tree: third moment matching, multiplicative approach) . . . 640
TimeBasket (Distribution over a number of dates) . . . 641
TimeGrid (Time grid class) . . . 642
Tokyo (Tokyo calendar) . . . 643
Toronto (Toronto calendar) . . . 645
TrapezoidIntegral (Integral of a one-dimensional function) . . . 646
Tree (**Tree**(p. 648) approximating a single-factor diffusion) . . . 648
TreeCapFloor (Cap/Floor priced in a tree) . . . 649
TreeSwaption (Swaption priced on a lattice) . . . 650
TridiagonalOperator (Base implementation for tridiagonal operator) . . . 651
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic) . . . 653
Trigeorgis (**Trigeorgis**(p. 654) (additive equal jumps) binomial tree) . . . 654
TrinomialBranching (Branching scheme for a trinomial node) . . . 655
TrinomialTree (Recombining trinomial tree class) . . . 656
TwoFactorModel (Abstract base-class for two-factor models) . . . 657
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables) . . . 658

TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable) . . . 659
UpFrontIndexedCoupon (up front indexed coupon class) . . . 660
USDLibor (USD Libor index) . . . 661
Value (Pricing results) . . . 662
VanillaEngine (Vanilla engine base class) . . . 663
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset) . . . 664
VanillaOption::arguments (**Arguments**(p. 169) for vanilla option calculation) . . . 667
VanillaOption::results (results from vanilla option calculation) . . . 668
Vasicek (**Vasicek**(p. 669) model class) . . . 669
Vasicek::Dynamics (Short-rate dynamics in the **Vasicek**(p. 669) model) . . . 671
Visitor (**Visitor**(p. 672) for a specific class) . . . 672
Warsaw (Warsaw calendar) . . . 673
Wellington (Wellington calendar) . . . 674
Xibor (Base class for libor indexes) . . . 675
XiborManager (Global repository for libor histories) . . . 677
ZARLibor (ZAR Libor index (also known as JIBAR)) . . . 678
ZeroCurve (Term structure based on linear interpolation of zero yields) . . . 679
ZeroSpreadedTermStructure (Term structure with an added spread on the zero yield rate) . . . 681
ZeroYieldStructure (Zero yield term structure) . . . 683
Zurich (Zurich calendar) . . . 685

Chapter 8

QuantLib File Index

8.1 QuantLib File List

Here is a list of all documented files with brief descriptions:

ql/**argsandresults.hpp** (Base classes for generic arguments and results) 687
ql/**array.hpp** (1-D array used in linear algebra) 688
ql/**blackmodel.hpp** (Abstract class for Black-type models (market models)) 689
ql/**calendar.cpp** (Abstract calendar class) 690
ql/**calendar.hpp** (calendar class) 691
ql/**capvolstructures.hpp** (Cap/Floor volatility structures) 727
ql/**cashflow.hpp** (Base class for cash flows) 728
ql/**currency.hpp** (Known currencies) 748
ql/**dataformatters.cpp** (Classes used to format data for output) 749
ql/**dataformatters.hpp** (Classes used to format data for output) 750
ql/**dataparsers.cpp** (Classes used to parse data for input) 751
ql/**dataparsers.hpp** (Classes used to parse data for input) 752
ql/**date.cpp** (Date- and time-related classes, typedefs and enumerations) 753
ql/**date.hpp** (Date- and time-related classes, typedefs and enumerations) 754
ql/**daycounter.hpp** (Day counter class) 755
ql/**diffusionprocess.cpp** (Diffusion process) 764
ql/**diffusionprocess.hpp** (Diffusion process) 765
ql/**discretizedasset.cpp** (Discretized asset classes) 766
ql/**discretizedasset.hpp** (Discretized asset classes) 767
ql/**disposable.hpp** (Generic disposable object with move semantics) 768
ql/**errors.hpp** (Classes and functions for error handling) 769
ql/**exercise.cpp** (Option exercise classes and exercise payoff function) 771
ql/**exercise.hpp** (Option exercise classes and payoff function) 772
ql/**grid.cpp** (Grid classes with useful constructors for trees and finite diffs) 802
ql/**grid.hpp** (Grid classes with useful constructors for trees and finite diffs) 803
ql/**handle.hpp** (Reference-counted pointer) 804
ql/**history.hpp** (History class) 805
ql/**index.hpp** (Purely virtual base class for indexes) 806
ql/**instrument.hpp** (Abstract instrument class) 819
ql/**marketelement.hpp** (Purely virtual base class for market observables) 853
ql/**null.hpp** (Null values) 936
ql/**numericalmethod.hpp** (Numerical method class) 937
ql/**option.hpp** (Base option class) 953

ql/**payoff.hpp** (Option payoff classes) . . . 959
ql/**pricingengine.hpp** (Base class for pricing engines) . . . 1033
ql/**qldefines.hpp** (Global definitions and compiler switches) . . . 1052
ql/**relinkablehandle.hpp** (Globally accessible relinkable pointer) . . . 1071
ql/**scheduler.cpp** (Date scheduler) . . . 1072
ql/**scheduler.hpp** (Date scheduler) . . . 1073
ql/**solver1d.hpp** (Abstract 1-D solver class) . . . 1099
ql/**swaptionvolstructure.hpp** (Swaption volatility structure) . . . 1107
ql/**termstructure.hpp** (Term structure) . . . 1108
ql/**types.hpp** (Custom types) . . . 1129
ql/**voltermstructure.cpp** (Volatility term structures) . . . 1148
ql/**voltermstructure.hpp** (Volatility term structures) . . . 1149
ql/Calendars/**budapest.cpp** (Budapest calendar) . . . 692
ql/Calendars/**budapest.hpp** (Budapest calendar) . . . 693
ql/Calendars/**frankfurt.cpp** (Frankfurt calendar) . . . 694
ql/Calendars/**frankfurt.hpp** (Frankfurt calendar) . . . 695
ql/Calendars/**helsinki.cpp** (Helsinki calendar) . . . 696
ql/Calendars/**helsinki.hpp** (Helsinki calendar) . . . 697
ql/Calendars/**johannesburg.cpp** (Johannesburg calendar) . . . 698
ql/Calendars/**johannesburg.hpp** (Johannesburg calendar) . . . 699
ql/Calendars/**jointcalendar.cpp** (Joint calendar) . . . 700
ql/Calendars/**jointcalendar.hpp** (Joint calendar) . . . 701
ql/Calendars/**london.cpp** (London calendar) . . . 702
ql/Calendars/**london.hpp** (London calendar) . . . 703
ql/Calendars/**milan.cpp** (Milan calendar) . . . 704
ql/Calendars/**milan.hpp** (Milan calendar) . . . 705
ql/Calendars/**newyork.cpp** (New York calendar) . . . 706
ql/Calendars/**newyork.hpp** (New York calendar) . . . 707
ql/Calendars/**nullcalendar.hpp** (Calendar for reproducing theoretical calculations) . . . 708
ql/Calendars/**oslo.cpp** (Oslo calendar) . . . 709
ql/Calendars/**oslo.hpp** (Oslo calendar) . . . 710
ql/Calendars/**stockholm.cpp** (Stockholm calendar) . . . 711
ql/Calendars/**stockholm.hpp** (Stockholm calendar) . . . 712
ql/Calendars/**sydney.cpp** (Sydney calendar) . . . 713
ql/Calendars/**sydney.hpp** (Sydney calendar) . . . 714
ql/Calendars/**target.cpp** (TARGET calendar) . . . 715
ql/Calendars/**target.hpp** (TARGET calendar) . . . 716
ql/Calendars/**tokyo.cpp** (Tokyo calendar) . . . 717
ql/Calendars/**tokyo.hpp** (Tokyo calendar) . . . 718
ql/Calendars/**toronto.cpp** (Toronto calendar) . . . 719
ql/Calendars/**toronto.hpp** (Toronto calendar) . . . 720
ql/Calendars/**warsaw.cpp** (Warsaw calendar) . . . 721
ql/Calendars/**warsaw.hpp** (Warsaw calendar) . . . 722
ql/Calendars/**wellington.cpp** (Wellington calendar) . . . 723
ql/Calendars/**wellington.hpp** (Wellington calendar) . . . 724
ql/Calendars/**zurich.cpp** (Zurich calendar) . . . 725
ql/Calendars/**zurich.hpp** (Zurich calendar) . . . 726
ql/CashFlows/**basispointsensitivity.cpp** (Basis point sensitivity calculator) . . . 729
ql/CashFlows/**basispointsensitivity.hpp** (Basis point sensitivity calculator) . . . 730
ql/CashFlows/**cashflowvectors.cpp** (Cash flow vector builders) . . . 731
ql/CashFlows/**cashflowvectors.hpp** (Cash flow vector builders) . . . 732
ql/CashFlows/**coupon.hpp** (Coupon accruing over a fixed period) . . . 733
ql/CashFlows/**fixedratecoupon.hpp** (Coupon paying a fixed annual rate) . . . 734
ql/CashFlows/**floatingratecoupon.hpp** (Coupon at par on a term structure) . . . 735

ql/CashFlows/**inarrearindexedcoupon.hpp** (In arrear indexed coupon) . . . 736
ql/CashFlows/**indexcashflowvectors.hpp** (Index Cash flow vector builders) . . . 737
ql/CashFlows/**indexedcoupon.hpp** (Indexed coupon) . . . 738
ql/CashFlows/**parcoupon.cpp** (Coupon at par on a term structure) . . . 739
ql/CashFlows/**parcoupon.hpp** (Coupon at par on a term structure) . . . 740
ql/CashFlows/**shortfloatingcoupon.cpp** (Short coupon at par on a term structure) . . . 741
ql/CashFlows/**shortfloatingcoupon.hpp** (Short (or long) coupon at par on a term structure) . . . 742
ql/CashFlows/**shortindexedcoupon.hpp** (Short (or long) indexed coupon) . . . 743
ql/CashFlows/**simplecashflow.hpp** (Predetermined cash flow) . . . 744
ql/CashFlows/**timebasket.cpp** . . . 745
ql/CashFlows/**timebasket.hpp** . . . 746
ql/CashFlows/**upfrontindexedcoupon.hpp** (Up front indexed coupon) . . . 747
ql/DayCounters/**actual360.hpp** (Act/360 day counter) . . . 756
ql/DayCounters/**actual365.hpp** (Act/365 day counter) . . . 757
ql/DayCounters/**actualactual.cpp** (Act/act day counters) . . . 758
ql/DayCounters/**actualactual.hpp** (Act/act day counters) . . . 759
ql/DayCounters/**simpledaycounter.cpp** (Simple day counter for reproducing theoretical calculations) . . . 760
ql/DayCounters/**simpledaycounter.hpp** (Simple day counter for reproducing theoretical calculations) . . . 761
ql/DayCounters/**thirty360.cpp** (30/360 day counters) . . . 762
ql/DayCounters/**thirty360.hpp** (30/360 day counters) . . . 763
ql/FiniteDifferences/**americancondition.hpp** (American option exercise condition) . . . 773
ql/FiniteDifferences/**boundarycondition.cpp** (Boundary conditions for differential operators) . . . 774
ql/FiniteDifferences/**boundarycondition.hpp** (Boundary conditions for differential operators) . . . 775
ql/FiniteDifferences/**bsmoperator.cpp** (Differential operator for Black-Scholes-Merton equation) . . . 776
ql/FiniteDifferences/**bsmoperator.hpp** (Differential operator for Black-Scholes-Merton equation) . . . 777
ql/FiniteDifferences/**cranknicolson.hpp** (Crank-Nicolson scheme for finite difference methods) . . . 778
ql/FiniteDifferences/**dminus.hpp** (D_- matricial representation) . . . 779
ql/FiniteDifferences/**dplus.hpp** (D_+ matricial representation) . . . 780
ql/FiniteDifferences/**dplusdminus.hpp** (D_+D_- matricial representation) . . . 781
ql/FiniteDifferences/**dzero.hpp** (D_0 matricial representation) . . . 782
ql/FiniteDifferences/**expliciteuler.hpp** (Explicit Euler scheme for finite difference methods) . . . 783
ql/FiniteDifferences/**fdtypedefs.hpp** (Default choices for template instantiations) . . . 784
ql/FiniteDifferences/**finitedifferencemodel.hpp** (Generic finite difference model) . . . 785
ql/FiniteDifferences/**impliciteuler.hpp** (Implicit Euler scheme for finite difference methods) . . . 786
ql/FiniteDifferences/**mixedscheme.hpp** (Mixed (explicit/implicit) scheme for finite difference methods) . . . 787
ql/FiniteDifferences/**onefactoroperator.cpp** (Differential operator for one-factor interest rate models) . . . 788
ql/FiniteDifferences/**onefactoroperator.hpp** (General differential operator for one-factor interest rate models) . . . 789
ql/FiniteDifferences/**shoutcondition.hpp** (Shout option exercise condition) . . . 790
ql/FiniteDifferences/**stepcondition.hpp** (Conditions to be applied at every time step) . . . 791
ql/FiniteDifferences/**tridiagonaloperator.cpp** (Tridiagonal operator) . . . 792
ql/FiniteDifferences/**tridiagonaloperator.hpp** (Tridiagonal operator) . . . 793

ql/FiniteDifferences/**valueatcenter.cpp** (Compute value, first, and second derivatives at grid center) . . . 794
ql/FiniteDifferences/**valueatcenter.hpp** (Compute value, first, and second derivatives at grid center) . . . 795
ql/functions/**daycounters.cpp** (Day counters functions) . . . 796
ql/functions/**daycounters.hpp** (Day counters functions) . . . 797
ql/functions/**mathf.cpp** (Math functions) . . . 798
ql/functions/**mathf.hpp** (Math functions) . . . 799
ql/functions/**vols.cpp** (Volatility functions) . . . 800
ql/functions/**vols.hpp** (Volatility functions) . . . 801
ql/Indexes/**audlibor.hpp** (AUD Libor index (check settlement days)) . . . 807
ql/Indexes/**cadlibor.hpp** (CAD Libor index (Also known as CDOR)) . . . 808
ql/Indexes/**chflibor.hpp** (CHF Libor index (Also known as ZIBOR)) . . . 809
ql/Indexes/**euribor.hpp** (Euribor index) . . . 810
ql/Indexes/**gbplibor.hpp** (GBP Libor index) . . . 811
ql/Indexes/**jpylibor.hpp** (JPY Libor index (Also known as TIBOR, check settlement days)) 812
ql/Indexes/**usdlibor.hpp** (USD Libor index) . . . 813
ql/Indexes/**xibor.cpp** (Purely virtual base class for libor indexes) . . . 814
ql/Indexes/**xibor.hpp** (Base class for libor indexes) . . . 815
ql/Indexes/**xibormanager.cpp** (Global repository for Xibor histories) . . . 816
ql/Indexes/**xibormanager.hpp** (Global repository for Xibor histories) . . . 817
ql/Indexes/**zarlibor.hpp** (ZAR Libor index (also known as JIBAR)) . . . 818
ql/Instruments/**barrieroption.cpp** (Barrier option on a single asset) . . . 820
ql/Instruments/**barrieroption.hpp** (Barrier option on a single asset) . . . 821
ql/Instruments/**binaryoption.cpp** (Binary option on a single asset) . . . 822
ql/Instruments/**binaryoption.hpp** (Binary option on a single asset) . . . 823
ql/Instruments/**capfloor.cpp** (European cap and floor class) . . . 824
ql/Instruments/**capfloor.hpp** (Cap and Floor class) . . . 825
ql/Instruments/**forwardvanillaoption.cpp** (Forward version of a vanilla option) . . . 826
ql/Instruments/**forwardvanillaoption.hpp** (Forward version of a vanilla option) . . . 827
ql/Instruments/**quantoforwardvanillaoption.cpp** (Quanto version of a forward vanilla option) . . . 828
ql/Instruments/**quantoforwardvanillaoption.hpp** (Quanto version of a forward vanilla option) . . . 829
ql/Instruments/**quantovanillaoption.cpp** (Quanto version of a vanilla option) . . . 830
ql/Instruments/**quantovanillaoption.hpp** (Quanto version of a vanilla option) . . . 831
ql/Instruments/**simpleswap.cpp** (Simple fixed-rate vs Libor swap) . . . 832
ql/Instruments/**simpleswap.hpp** (Simple fixed-rate vs Libor swap) . . . 833
ql/Instruments/**stock.cpp** (Concrete stock class) . . . 834
ql/Instruments/**stock.hpp** (Concrete stock class) . . . 835
ql/Instruments/**swap.cpp** (Interest rate swap) . . . 836
ql/Instruments/**swap.hpp** (Interest rate swap) . . . 837
ql/Instruments/**swaption.cpp** (Swaption) . . . 838
ql/Instruments/**swaption.hpp** (Swaption class) . . . 839
ql/Instruments/**vanillaoption.cpp** (Vanilla option on a single asset) . . . 840
ql/Instruments/**vanillaoption.hpp** (Vanilla option on a single asset) . . . 841
ql/Lattices/**binomialtree.cpp** (Binomial tree class) . . . 842
ql/Lattices/**binomialtree.hpp** (Binomial tree class) . . . 843
ql/Lattices/**bsmlattice.cpp** (Binomial trees under the BSM model) . . . 844
ql/Lattices/**bsmlattice.hpp** (Binomial trees under the BSM model) . . . 845
ql/Lattices/**lattice.cpp** (Lattice class) . . . 846
ql/Lattices/**lattice.hpp** (Lattice method class) . . . 847
ql/Lattices/**lattice2d.cpp** (Trinomial tree class) . . . 848
ql/Lattices/**lattice2d.hpp** (Two-dimensional tree class) . . . 849

ql/Lattices/**tree.hpp** (Tree class) . . . 850
ql/Lattices/**trinomialtree.cpp** (Trinomial tree class) . . . 851
ql/Lattices/**trinomialtree.hpp** (Trinomial tree class) . . . 852
ql/Math/**bicubicsplineinterpolation.hpp** (Bicubic spline interpolation between discrete points) . . . 854
ql/Math/**bilinearinterpolation.hpp** (Bilinear interpolation between discrete points) . . . 855
ql/Math/**chisquaredistribution.cpp** (Normal, cumulative and inverse cumulative distributions) . . . 856
ql/Math/**chisquaredistribution.hpp** (Chi-square (central and non-central) distributions) 857
ql/Math/**cubicspline.hpp** (Cubic spline interpolation between discrete points) . . . 858
ql/Math/**discrepancystatistics.cpp** (Statistic tool for sequences with discrepancy calculation) . . . 859
ql/Math/**discrepancystatistics.hpp** (Statistic tool for sequences with discrepancy calculation) . . . 860
ql/Math/**errorfunction.cpp** (Error function) . . . 861
ql/Math/**errorfunction.hpp** (Error function) . . . 862
ql/Math/**functional.hpp** (Functionals and combinators not included in the STL) . . . 863
ql/Math/**gammadistribution.cpp** (Gamma function) . . . 864
ql/Math/**gammadistribution.hpp** (Gamma distribution) . . . 865
ql/Math/**gaussianstatistics.hpp** (Statistics tool for gaussian-assumption risk measures) 866
ql/Math/**generalstatistics.cpp** (Statistics tool) . . . 867
ql/Math/**generalstatistics.hpp** (Statistics tool) . . . 868
ql/Math/**incrementalstatistics.cpp** (Statistics tool based on incremental accumulation) . 869
ql/Math/**incrementalstatistics.hpp** (Statistics tool based on incremental accumulation) . 870
ql/Math/**interpolation.hpp** (Abstract base classes for interpolations) . . . 871
ql/Math/**interpolation2D.hpp** (Abstract base classes for 2-D interpolations) . . . 872
ql/Math/**interpolationtraits.hpp** (Traits classes for interpolation algorithms) . . . 873
ql/Math/**kronrodintegral.hpp** (Integral of a 1-dimensional function using the Gauss-Kronrod method) . . . 874
ql/Math/**lexicographicalview.hpp** (Lexicographical 2-D view of a contiguous set of data) 875
ql/Math/**linearinterpolation.hpp** (Linear interpolation between discrete points) . . . 876
ql/Math/**loglinearinterpolation.hpp** (Log-linear interpolation between discrete points) 877
ql/Math/**matrix.cpp** (Matrix used in linear algebra) . . . 878
ql/Math/**matrix.hpp** (Matrix used in linear algebra) . . . 879
ql/Math/**multivariateaccumulator.cpp** (A simple accumulator for vector-type samples) 880
ql/Math/**multivariateaccumulator.hpp** (A simple accumulator for vector-type samples) 881
ql/Math/**normaldistribution.cpp** (Normal, cumulative and inverse cumulative distributions) . . . 882
ql/Math/**normaldistribution.hpp** (Normal, cumulative and inverse cumulative distributions) . . . 883
ql/Math/**primenumbers.cpp** (Prime numbers calculator) . . . 884
ql/Math/**primenumbers.hpp** (Prime numbers calculator) . . . 885
ql/Math/**riskmeasures.hpp** (Risk functions) . . . 886
ql/Math/**segmentintegral.hpp** (Integral of a one-dimensional function) . . . 887
ql/Math/**sequencestatistics.hpp** (Statistics tools for sequence (vector, list, array) samples) 888
ql/Math/**statistics.hpp** (Statistics tool with risk measures) . . . 890
ql/Math/**svd.cpp** (Calculates Singular Value Decomposition) . . . 891
ql/Math/**svd.hpp** (Calculates Singular Value Decomposition) . . . 892
ql/Math/**symmetriceigenvalues.hpp** (Eigenvalues / eigenvectors of a real symmetric matrix) . . . 893
ql/Math/**symmetricschurdecomposition.cpp** (Eigenvalues / eigenvectors of a real symmetric matrix) . . . 894
ql/Math/**symmetricschurdecomposition.hpp** (Eigenvalues / eigenvectors of a real symmetric matrix) . . . 895

ql/Math/**trapezoidintegral.hpp** (Integral of a one-dimensional function) 896
ql/MonteCarlo/**arithmeticapopathpricer.cpp** (Arithmetic average price option path pricer) 897
ql/MonteCarlo/**arithmeticapopathpricer.hpp** (Arithmetic average price option path pricer) 898
ql/MonteCarlo/**arithmeticasopathpricer.cpp** (Arithmetic average strike option path pricer) 899
ql/MonteCarlo/**arithmeticasopathpricer.hpp** (Arithmetic average strike option path pricer) 900
ql/MonteCarlo/**barrierpathpricer.cpp** (Path pricer for Barrier options) 901
ql/MonteCarlo/**barrierpathpricer.hpp** (Path pricer for Barrier options) 902
ql/MonteCarlo/**basketpathpricer.cpp** (Multipath pricer for European-type basket option) 903
ql/MonteCarlo/**basketpathpricer.hpp** (Multipath pricer for European-type basket option) 904
ql/MonteCarlo/**binarypathpricer.cpp** (Path pricer for Binary options) 905
ql/MonteCarlo/**binarypathpricer.hpp** (Path pricer for Binary options) 906
ql/MonteCarlo/**brownianbridge.hpp** (Browian bridge) 907
ql/MonteCarlo/**cliquetoptionpathpricer.cpp** (Path pricer for cliquet options) 908
ql/MonteCarlo/**cliquetoptionpathpricer.hpp** (Path pricer for cliquet options) 909
ql/MonteCarlo/**europeanpathpricer.cpp** (Path pricer for European options) 910
ql/MonteCarlo/**europeanpathpricer.hpp** (Path pricer for European options) 911
ql/MonteCarlo/**everestpathpricer.cpp** (Path pricer for European-type Everest option) . 912
ql/MonteCarlo/**everestpathpricer.hpp** (Path pricer for European-type Everest option) . 913
ql/MonteCarlo/**geometricapopathpricer.cpp** (Path pricer for geometric average price option) 914
ql/MonteCarlo/**geometricapopathpricer.hpp** (Path pricer for geometric average price option) 915
ql/MonteCarlo/**geometricasopathpricer.cpp** (Path pricer for geometric average strike option) 916
ql/MonteCarlo/**geometricasopathpricer.hpp** (Path pricer for geometric average strike option) 917
ql/MonteCarlo/**getcovariance.hpp** (Covariance matrix calculation) 918
ql/MonteCarlo/**himalayapathpricer.cpp** (Multipath pricer for European-type Himalaya option) 919
ql/MonteCarlo/**himalayapathpricer.hpp** (Multipath pricer for European-type Himalaya option) 920
ql/MonteCarlo/**maxbasketpathpricer.cpp** (Multipath pricer for max basket option) . . . 921
ql/MonteCarlo/**maxbasketpathpricer.hpp** (Multipath pricer for max basket option) . . 922
ql/MonteCarlo/**mctraits.hpp** (Monte Carlo policy descriptors) 923
ql/MonteCarlo/**mctypedefs.hpp** (Default choices for template instantiations) 924
ql/MonteCarlo/**montecarlomodel.hpp** (General purpose Monte Carlo model) 925
ql/MonteCarlo/**multipath.hpp** (Correlated multiple asset paths) 926
ql/MonteCarlo/**multipathgenerator.hpp** (Generates a multi path from a random-array generator) 927
ql/MonteCarlo/**pagodapathpricer.cpp** (Path pricer for pagoda options) 928
ql/MonteCarlo/**pagodapathpricer.hpp** (Path pricer for pagoda options) 929
ql/MonteCarlo/**path.hpp** (Single factor random walk) 930
ql/MonteCarlo/**pathgenerator.hpp** (Generates random paths using a sequence generator) 931
ql/MonteCarlo/**pathpricer.hpp** (Base class for single-path pricers) 932
ql/MonteCarlo/**performanceoptionpathpricer.cpp** (Path pricer for performance options) 933
ql/MonteCarlo/**performanceoptionpathpricer.hpp** (Path pricer for performance option) 934
ql/MonteCarlo/**sample.hpp** (Weighted sample) 935
ql/Optimization/**armijo.cpp** (Armijo line-search class) 938
ql/Optimization/**armijo.hpp** (Armijo line-search class) 939
ql/Optimization/**conjugategradient.cpp** (Conjugate gradient optimization method) . . 940

ql/Optimization/**conjugategradient.hpp** (Conjugate gradient optimization method) . . 941
ql/Optimization/**constraint.hpp** (Abstract constraint class) 942
ql/Optimization/**costfunction.hpp** (Optimization cost function class) 943
ql/Optimization/**criteria.hpp** (Optimization criteria class) 944
ql/Optimization/**leastsquare.hpp** (Least square cost function) 945
ql/Optimization/**linesearch.hpp** (Line search abstract class) 946
ql/Optimization/**method.hpp** (Abstract optimization method class) 947
ql/Optimization/**problem.hpp** (Abstract optimization class) 948
ql/Optimization/**simplex.cpp** (Simplex optimization method) 949
ql/Optimization/**simplex.hpp** (Simplex optimization method) 950
ql/Optimization/**steepestdescent.cpp** (Steepest descent optimization method) 951
ql/Optimization/**steepestdescent.hpp** (Steepest descent optimization method) 952
ql/Patterns/**bridge.hpp** (Bridge pattern (a.k.a. handle-body idiom)) 954
ql/Patterns/**curiouslyrecurring.hpp** (Curiously recurring template pattern) 955
ql/Patterns/**lazyobject.hpp** (Framework for calculation on demand and result caching) 956
ql/Patterns/**observable.hpp** (Observer/observable pattern) 957
ql/Patterns/**visitor.hpp** (Degenerate base class for the Acyclic Visitor pattern) 958
ql/Pricers/**analyticalcapfloor.cpp** (Analytical pricer for caps/floors) 960
ql/Pricers/**analyticalcapfloor.hpp** (Analytical pricer for caps/floors) 961
ql/Pricers/**barrieroptionpricer.cpp** (Barrier option pricer) 962
ql/Pricers/**barrieroptionpricer.hpp** (Barrier option pricer) 963
ql/Pricers/**binaryoptionpricer.cpp** (European style cash-or-nothing option pricer) . . . 964
ql/Pricers/**binaryoptionpricer.hpp** (European style cash-or-nothing option pricer) . . . 965
ql/Pricers/**blackcapfloor.cpp** (European capfloor calculated using Black formula) 966
ql/Pricers/**blackcapfloor.hpp** (CapFloor calculated using the Black formula) 967
ql/Pricers/**blackswaption.cpp** (European swaption calculated using Black formula) . . 968
ql/Pricers/**blackswaption.hpp** (Swaption calculated using the Black formula) 969
ql/Pricers/**capfloorpricer.cpp** (Cap and floor pricer class) 970
ql/Pricers/**capfloorpricer.hpp** (Cap and floor pricer class) 971
ql/Pricers/**cliquetoption.cpp** (Cliquet option) 972
ql/Pricers/**continuousgeometricapo.hpp** (Continuous Geometric Average Price Option (European exercise)) 973
ql/Pricers/**discretegeometricapo.cpp** (Discrete Geometric Average Price Option) 974
ql/Pricers/**discretegeometricapo.hpp** (Discrete Geometric Average Price Option) 975
ql/Pricers/**discretegeometricaso.cpp** (Discrete Geometric Average Strike Option) 976
ql/Pricers/**discretegeometricaso.hpp** (Discrete Geometric Average Strike Option) 977
ql/Pricers/**europeanoption.cpp** (European option) 978
ql/Pricers/**europeanoption.hpp** (European option) 979
ql/Pricers/**fdamericanoption.hpp** (American option) 980
ql/Pricers/**fdbermudanoption.cpp** (Finite-difference evaluation of Bermudan option) . 981
ql/Pricers/**fdbermudanoption.hpp** (Finite-difference evaluation of Bermudan option) . 982
ql/Pricers/**fdbsmoption.cpp** (Common code for numerical option evaluation) 983
ql/Pricers/**fdbsmoption.hpp** (Common code for numerical option evaluation) 984
ql/Pricers/**fddividendamericanoption.cpp** (American option with discrete deterministic dividends) 985
ql/Pricers/**fddividendamericanoption.hpp** (American option with discrete deterministic dividends) 986
ql/Pricers/**fddividendeuropeanoption.cpp** (European option with discrete deterministic dividends) 987
ql/Pricers/**fddividendeuropeanoption.hpp** (European option with discrete deterministic dividends) 988
ql/Pricers/**fddividendoption.cpp** (Base class for options with dividends) 989
ql/Pricers/**fddividendoption.hpp** (Base class for option with dividends) 990
ql/Pricers/**fddividendshoutoption.cpp** (Base class for shout options with dividends) . . 991

ql/Pricers/**fddividendshoutoption.hpp** (Base class for shout option with dividends) . . 992
ql/Pricers/**fdeuropean.cpp** (Example of European option calculated using finite differences) . . . 993
ql/Pricers/**fdeuropean.hpp** (Example of European option calculated using finite differences) . . . 994
ql/Pricers/**fdmultiperiodoption.cpp** (Base class for option with events happening at different periods) . . . 995
ql/Pricers/**fdmultiperiodoption.hpp** (Base class for option with events happening at different periods) . . . 996
ql/Pricers/**fdshoutoption.hpp** (Shout option) . . . 997
ql/Pricers/**fdstepconditionoption.cpp** (Option requiring additional code to be executed at each time step) . . . 998
ql/Pricers/**fdstepconditionoption.hpp** (Option requiring additional code to be executed at each time step) . . . 999
ql/Pricers/**jamshidianswaption.cpp** (Swaption pricer using Jamshidian's decomposition)1000
ql/Pricers/**jamshidianswaption.hpp** (Swaption pricer using Jamshidian's decomposition) . . . 1001
ql/Pricers/**mcbasket.cpp** (Simple example of multi-factor Monte Carlo pricer) . . . 1002
ql/Pricers/**mcbasket.hpp** (Simple example of multi-factor Monte Carlo pricer) . . . 1003
ql/Pricers/**mccliquetoption.cpp** (Cliquet option priced with Monte Carlo simulation) . 1004
ql/Pricers/**mccliquetoption.hpp** (Cliquet option priced with Monte Carlo simulation) . 1005
ql/Pricers/**mcdiscretearithmeticapo.cpp** (Discrete Arithmetic Average Price Option) . . 1006
ql/Pricers/**mcdiscretearithmeticapo.hpp** (Discrete Arithmetic Average Price Option) . . 1007
ql/Pricers/**mcdiscretearithmeticaso.cpp** (Discrete Arithmetic Average Strike Option) . . 1008
ql/Pricers/**mcdiscretearithmeticaso.hpp** (Discrete Arithmetic Average Strike Option) . 1009
ql/Pricers/**mceuropean.cpp** (Simple example of Monte Carlo pricer) . . . 1010
ql/Pricers/**mceuropean.hpp** (Simple example of Monte Carlo pricer) . . . 1011
ql/Pricers/**mceverest.cpp** (Everest-type option pricer) . . . 1012
ql/Pricers/**mceverest.hpp** (Everest-type option pricer) . . . 1013
ql/Pricers/**mchimalaya.cpp** (Himalayan-type option pricer) . . . 1014
ql/Pricers/**mchimalaya.hpp** (Himalayan-type option pricer) . . . 1015
ql/Pricers/**mcmaxbasket.cpp** (Max Basket Monte Carlo pricer) . . . 1016
ql/Pricers/**mcmaxbasket.hpp** (Max Basket Monte Carlo pricer) . . . 1017
ql/Pricers/**mcpagoda.cpp** (Roofed multi asset Asian option) . . . 1018
ql/Pricers/**mcpagoda.hpp** (Roofed multi asset Asian option) . . . 1019
ql/Pricers/**mcperformanceoption.cpp** (Performance option priced with Monte Carlo simulation) . . . 1020
ql/Pricers/**mcperformanceoption.hpp** (Performance option priced with Monte Carlo simulation) . . . 1021
ql/Pricers/**mcpricer.hpp** (Base class for Monte Carlo pricers) . . . 1022
ql/Pricers/**performanceoption.cpp** (Performance option.computed using Monte Carlo simulation) . . . 1023
ql/Pricers/**performanceoption.hpp** (Performance option) . . . 1024
ql/Pricers/**singleassetoption.cpp** (Common code for option evaluation) . . . 1025
ql/Pricers/**singleassetoption.hpp** (Common code for option evaluation) . . . 1026
ql/Pricers/**swaptionpricer.cpp** (Swaption pricer) . . . 1027
ql/Pricers/**swaptionpricer.hpp** (Swaption pricer class) . . . 1028
ql/Pricers/**treecapfloor.cpp** (Cap/Floor calculated using a tree) . . . 1029
ql/Pricers/**treecapfloor.hpp** (Cap/Floor calculated using a tree) . . . 1030
ql/Pricers/**treeswaption.cpp** (European swaption computed on a lattice) . . . 1031
ql/Pricers/**treeswaption.hpp** (Swaption computed using a lattice) . . . 1032
ql/PricingEngines/**americanmcengines.cpp** (Least-square Monte Carlo engines) . . . 1034
ql/PricingEngines/**americanmcengines.hpp** (Least-square Monte Carlo engines) . . . 1035

ql/PricingEngines/**analyticamericanbinaryengine.cpp** (American binary option engine using analytic formulas) . . . 1036
ql/PricingEngines/**analyticeuropeanbinaryengine.cpp** (European binary option engine) 1037
ql/PricingEngines/**analyticeuropeanengine.cpp** (European option engine using analytic formulas) . . . 1038
ql/PricingEngines/**barrierengines.hpp** (Barrier option engines) . . . 1039
ql/PricingEngines/**binaryengines.hpp** (Binary option engines) . . . 1040
ql/PricingEngines/**binomialvanillaengine.cpp** (Vanilla option engine using binomial tree) . . . 1041
ql/PricingEngines/**cliquetengines.hpp** (Cliquet option engines) . . . 1042
ql/PricingEngines/**discretizedvanillaoption.cpp** (Discretized vanilla option) . . . 1043
ql/PricingEngines/**discretizedvanillaoption.hpp** (Discretized vanilla option) . . . 1044
ql/PricingEngines/**forwardengines.hpp** (Forward (strike-resetting) option engines) . . . 1045
ql/PricingEngines/**genericengine.hpp** (Generic option engine) . . . 1046
ql/PricingEngines/**integralengines.cpp** (Option engines using the integral approach) . . . 1047
ql/PricingEngines/**latticeshortratemodelengine.hpp** (Engine for a short-rate model specialized on a lattice) . . . 1048
ql/PricingEngines/**mcengine.hpp** (Base class for Monte Carlo engines) . . . 1049
ql/PricingEngines/**quantoengines.hpp** (Quanto option engines) . . . 1050
ql/PricingEngines/**vanillaengines.hpp** (Vanilla option engines) . . . 1051
ql/RandomNumbers/**boxmullergaussianrng.hpp** (Box-Muller Gaussian random-number generator) . . . 1054
ql/RandomNumbers/**centrallimitgaussianrng.hpp** (Central limit Gaussian random-number generator) . . . 1055
ql/RandomNumbers/**haltonrsg.cpp** (Halton low-discrepancy sequence generator) . . . 1056
ql/RandomNumbers/**haltonrsg.hpp** (Halton low-discrepancy sequence generator) . . . 1057
ql/RandomNumbers/**inversecumgaussianrng.hpp** (Inverse cumulative Gaussian random-number generator) . . . 1058
ql/RandomNumbers/**inversecumgaussianrsg.hpp** (Inverse cumulative Gaussian random sequence generator) . . . 1059
ql/RandomNumbers/**knuthuniformrng.cpp** (Knuth uniform random number generator) 1060
ql/RandomNumbers/**knuthuniformrng.hpp** (Knuth uniform random number generator)1061
ql/RandomNumbers/**lecuyeruniformrng.cpp** (L'Ecuyer uniform random number generator) . . . 1062
ql/RandomNumbers/**lecuyeruniformrng.hpp** (L'Ecuyer uniform random number generator) . . . 1063
ql/RandomNumbers/**mt19937uniformrng.cpp** (Mersenne Twister uniform random number generator) . . . 1064
ql/RandomNumbers/**mt19937uniformrng.hpp** (Mersenne Twister uniform random number generator) . . . 1065
ql/RandomNumbers/**randomarraygenerator.hpp** (Generates random arrays from a random number generator) . . . 1066
ql/RandomNumbers/**randomsequencegenerator.hpp** (Random sequence generator based on a pseudo-random number generator) . . . 1067
ql/RandomNumbers/**rngtypedefs.hpp** (Default choices for template instantiations) . . . 1068
ql/RandomNumbers/**sobolrsg.cpp** (Sobol low-discrepancy sequence generator) . . . 1069
ql/RandomNumbers/**sobolrsg.hpp** (Sobol low-discrepancy sequence generator) . . . 1070
ql/ShortRateModels/**calibrationhelper.cpp** (Calibration helper class) . . . 1074
ql/ShortRateModels/**calibrationhelper.hpp** (Calibration helper class) . . . 1075
ql/ShortRateModels/**model.cpp** (Abstract interest rate model class) . . . 1080
ql/ShortRateModels/**model.hpp** (Abstract interest rate model class) . . . 1081
ql/ShortRateModels/**onefactormodel.cpp** (Abstract one-factor interest rate model class) 1082
ql/ShortRateModels/**onefactormodel.hpp** (Abstract one-factor interest rate model class) 1083
ql/ShortRateModels/**parameter.hpp** (Model parameter classes) . . . 1094

ql/ShortRateModels/**twofactormodel.cpp** (Abstract two-factor interest rate model class) 1095
ql/ShortRateModels/**twofactormodel.hpp** (Abstract two-factor interest rate model class) 1096
ql/ShortRateModels/CalibrationHelpers/**caphelper.cpp** (Cap calibration helper) 1076
ql/ShortRateModels/CalibrationHelpers/**caphelper.hpp** (CapHelper calibration helper) 1077
ql/ShortRateModels/CalibrationHelpers/**swaptionhelper.cpp** (Swaption calibration helper) 1078
ql/ShortRateModels/CalibrationHelpers/**swaptionhelper.hpp** (Swaption calibration helper) 1079
ql/ShortRateModels/OneFactorModels/**blackkarasinski.cpp** (Black-Karasinski model) . 1084
ql/ShortRateModels/OneFactorModels/**blackkarasinski.hpp** (Black-Karasinski model) 1085
ql/ShortRateModels/OneFactorModels/**coxingersollross.cpp** (Cox-Ingersoll-Ross model) 1086
ql/ShortRateModels/OneFactorModels/**coxingersollross.hpp** (Cox-Ingersoll-Ross model) 1087
ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.cpp** (Extended Cox-Ingersoll-Ross model) 1088
ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.hpp** (Extended Cox-Ingersoll-Ross model) 1089
ql/ShortRateModels/OneFactorModels/**hullwhite.cpp** (Hull & White model) 1090
ql/ShortRateModels/OneFactorModels/**hullwhite.hpp** (Hull & White (HW) model) . . 1091
ql/ShortRateModels/OneFactorModels/**vasicek.cpp** (Vasicek model class) 1092
ql/ShortRateModels/OneFactorModels/**vasicek.hpp** (Vasicek model class) 1093
ql/ShortRateModels/TwoFactorModels/**g2.cpp** (Two-factor additive Gaussian Model G2++) 1097
ql/ShortRateModels/TwoFactorModels/**g2.hpp** (Two-factor additive Gaussian Model G2++) 1098
ql/Solvers1D/**bisection.hpp** (Bisection 1-D solver) 1100
ql/Solvers1D/**brent.hpp** (Brent 1-D solver) 1101
ql/Solvers1D/**falseposition.hpp** (False-position 1-D solver) 1102
ql/Solvers1D/**newton.hpp** (Newton 1-D solver) 1103
ql/Solvers1D/**newtonsafe.hpp** (Safe (bracketed) Newton 1-D solver) 1104
ql/Solvers1D/**ridder.hpp** (Ridder 1-D solver) 1105
ql/Solvers1D/**secant.hpp** (Secant 1-D solver) 1106
ql/TermStructures/**affinetermstructure.cpp** (Affine term-structure) 1109
ql/TermStructures/**affinetermstructure.hpp** (Affine term structure) 1110
ql/TermStructures/**compoundforward.cpp** (Compounded forward term structure) . . . 1111
ql/TermStructures/**compoundforward.hpp** (Compounded forward term structure) . . . 1112
ql/TermStructures/**discountcurve.cpp** (Term structure based on loglinear interpolation of discount factors) 1113
ql/TermStructures/**discountcurve.hpp** (Pre-bootstrapped discount factor structure) . . . 1114
ql/TermStructures/**drifttermstructure.hpp** (Drift term structure) 1115
ql/TermStructures/**extendeddiscountcurve.cpp** (Discount factor structure with detailed compound-forward calculation) 1116
ql/TermStructures/**extendeddiscountcurve.hpp** (Discount factor structure with detailed compound-forward calculation) 1117
ql/TermStructures/**flatforward.hpp** (Flat forward rate term structure) 1118
ql/TermStructures/**forwardspreadedtermstructure.hpp** (Forward spreaded term structure) 1119
ql/TermStructures/**impliedtermstructure.hpp** (Implied term structure) 1120
ql/TermStructures/**piecewiseflatforward.cpp** (Piecewise flat forward term structure) . . 1121
ql/TermStructures/**piecewiseflatforward.hpp** (Piecewise flat forward term structure) . . 1122
ql/TermStructures/**quantotermstructure.hpp** (Quanto term structure) 1123
ql/TermStructures/**ratehelpers.cpp** (Rate helpers base class) 1124
ql/TermStructures/**ratehelpers.hpp** (Rate helpers base class) 1125

ql/TermStructures/**zerocurve.cpp** (Term structure based on linear interpolation of zero yields) . . . 1126
ql/TermStructures/**zerocurve.hpp** (Pre-bootstrapped zero curve structure) . . . 1127
ql/TermStructures/**zerospreadedtermstructure.hpp** (Zero spreaded term structure) . . . 1128
ql/Utilities/**combiningiterator.hpp** (Iterator mapping a function to a set of underlying sequences) . . . 1130
ql/Utilities/**couplingiterator.hpp** (Iterator mapping a function to a pair of underlying sequences) . . . 1131
ql/Utilities/**filteringiterator.hpp** (Iterator filtering undesired data) . . . 1132
ql/Utilities/**iteratorcategories.hpp** (Lowest common denominator between two iterator categories) . . . 1133
ql/Utilities/**processingiterator.hpp** (Iterator mapping a unary function to an underlying sequence) . . . 1134
ql/Utilities/**steppingiterator.hpp** (Iterator advancing in constant steps) . . . 1135
ql/Volatilities/**blackconstantvol.hpp** (Black constant volatility, no time dependence, no strike dependence) . . . 1136
ql/Volatilities/**blackvariancecurve.cpp** (Black volatility curve modelled as variance curve) 1137
ql/Volatilities/**blackvariancecurve.hpp** (Black volatility curve modelled as variance curve) . . . 1138
ql/Volatilities/**blackvariancesurface.cpp** (Black volatility surface modelled as variance surface) . . . 1139
ql/Volatilities/**blackvariancesurface.hpp** (Black volatility surface modelled as variance surface) . . . 1140
ql/Volatilities/**capflatvolvector.hpp** (Cap/floor at-the-money flat volatility vector) . . . 1141
ql/Volatilities/**impliedvoltermstructure.hpp** (Implied Black Vol Term Structure) . . . 1142
ql/Volatilities/**localconstantvol.hpp** (Local constant volatility, no time dependence, no asset dependence) . . . 1143
ql/Volatilities/**localvolcurve.hpp** (Local volatility curve derived from a Black curve) . . . 1144
ql/Volatilities/**localvolsurface.cpp** (Local volatility surface derived from a Black vol surface) . . . 1145
ql/Volatilities/**localvolsurface.hpp** (Local volatility surface derived from a Black vol surface) . . . 1146
ql/Volatilities/**swaptionvolmatrix.hpp** (Swaption at-the-money volatility matrix) . . . 1147

Chapter 9

QuantLib Module Documentation

9.1 Global QuantLib macros

9.1.1 Detailed Description

Global definitions and quite a few macros which help porting the code to different compilers

Defines

- #define **QL_HEX_VERSION** 0x000304f0

  version hexadecimal number

- #define **QL_VERSION** "0.3.4"

  version string

- #define **QL_DUMMY_RETURN**(x)

  specific per-compiler definitions Is a dummy return statement required?

- #define **QL_IO_INIT** std::cout << std::string();

  I/O initialization.

9.1.2 Define Documentation

9.1.2.1 #define QL_DUMMY_RETURN(x)

specific per-compiler definitions Is a dummy return statement required?

Some compilers will issue a warning if it is missing even though it could never be reached during execution, e.g., after a block like

```
if (condition)
    return validResult;
else
    throw HideousError();
```

On the other hand, other compilers will issue a warning if it is present because it cannot be reached. For the code to be portable this macro should be used after the block.

9.1.2.2 #define QL_IO_INIT std::cout << std::string();

I/O initialization.

Sometimes, programs compiled with the free Borland compiler will crash miserably upon attempting to write on std::cout. Strangely enough, issuing the instruction

```
std::cout << std::string();
```

at the beginning of the program will prevent other accesses to std::cout from crashing the program. This macro, to be called at the beginning of main(), encapsulates the above enchantment for Borland and is defined as empty for the other compilers.

9.2 Math functions

9.2.1 Detailed Description

Some compilers still define math functions them in the global namespace. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_SQRT** std::sqrt
  square root
- #define **QL_FABS** std::fabs
  absolute value
- #define **QL_EXP** std::exp
  exponential
- #define **QL_LOG** std::log
  logarithm
- #define **QL_SIN** std::sin
  sine
- #define **QL_COS** std::cos
  cosine
- #define **QL_POW** std::pow
  power
- #define **QL_MODF** std::modf
  floating-point module

9.3 Numeric limits

9.3.1 Detailed Description

Some compilers do not give an implementation of yet. For the code to be portable these macros should be used instead of the corresponding method of std::numeric_limits or the corresponding macro defined in <limits.h>.

Defines

- #define **QL_MIN_INT** ((std::numeric_limits<int>::min)())
- #define **QL_MAX_INT** ((std::numeric_limits<int>::max)())
- #define **QL_MIN_DOUBLE** -((std::numeric_limits<double>::max)())
- #define **QL_MIN_POSITIVE_DOUBLE** ((std::numeric_limits<double>::min)())
- #define **QL_MAX_DOUBLE** ((std::numeric_limits<double>::max)())
- #define **QL_EPSILON** ((std::numeric_limits<double>::epsilon)())

9.3.2 Define Documentation

9.3.2.1 #define QL_MIN_INT ((std::numeric_limits<int>::min)())

Defines the value of the largest representable negative integer value

9.3.2.2 #define QL_MAX_INT ((std::numeric_limits<int>::max)())

Defines the value of the largest representable integer value

9.3.2.3 #define QL_MIN_DOUBLE -((std::numeric_limits<double>::max)())

Defines the value of the largest representable negative double value

9.3.2.4 #define QL_MIN_POSITIVE_DOUBLE ((std::numeric_limits<double>::min)())

Defines the value of the smallest representable positive double value

9.3.2.5 #define QL_MAX_DOUBLE ((std::numeric_limits<double>::max)())

Defines the value of the largest representable double value

9.3.2.6 #define QL_EPSILON ((std::numeric_limits<double>::epsilon)())

Defines the machine precision for operations over doubles

9.4 Time functions

9.4.1 Detailed Description

Some compilers still define time functions in the global namespace. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_CLOCK** std::clock

  clock value

- #define **QL_TIME** std::time

  time value

9.5 String functions

9.5.1 Detailed Description

Some compilers still define string functions in the global namespace. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_STRLEN** std::strlen

  string length

9.6 Character functions

9.6.1 Detailed Description

Some compilers still define character functions in the global namespace. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_TOUPPER** std::toupper

  convert to uppercase

- #define **QL_TOLOWER** std::tolower

  convert to lowercase

9.7 Input/output functions

9.7.1 Detailed Description

Some compilers still define i/o functions in the global namespace. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_SPRINTF** std::sprintf

  print to string

9.8 Min and max functions

9.8.1 Detailed Description

Some compilers still do not define std::min and std::max. Moreover, Visual C++ defines them but for unfathomable reasons garble their names. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_MIN** std::min

  minimum between two elements

- #define **QL_MAX** std::max

  maximum between two elements

9.9 Template capabilities

9.9.1 Detailed Description

Some compilers still do not fully implement the template syntax. These macros can be used to select between alternate implementations of blocks of code, namely, one that takes advantage of template programming techniques and a less efficient one which is compatible with all compilers.

Defines

- #define **QL_DECLARE_TEMPLATE_SPECIALIZATIONS**

  Blame Microsoft for this one...

- #define **QL_ALLOW_TEMPLATE_METHOD_CALLS** 1

  Blame Microsoft for this one...

- #define **QL_TYPENAME** typename

  Blame Microsoft for this one...

- #define **QL_TEMPLATE_METAPROGRAMMING_WORKS** 1

9.9.2 Define Documentation

9.9.2.1 #define QL_DECLARE_TEMPLATE_SPECIALIZATIONS

Blame Microsoft for this one...

They decided that a declaration and a definition of a specialized template function amount to a redefinition and should issue a linker error. For the code to be portable, template specializations should be declared (as opposed to defined) only if this macro is defined.

9.9.2.2 #define QL_ALLOW_TEMPLATE_METHOD_CALLS 1

Blame Microsoft for this one...

Their compiler cannot cope with method calls such as

```
Handle<Type1> h1(whatever);
h2 = h1.downcast<Type2>();
```

For compatibility, a workaround should be implemented (which of course will be less solid or more comples - as I said, blame Microsoft...)

9.9.2.3 #define QL_TYPENAME typename

Blame Microsoft for this one...

They decided that typename can only be used in template declarations and not in template definitions.

9.9.2.4 #define QL_TEMPLATE_METAPROGRAMMING_WORKS 1

Template metaprogramming techniques (see T. L. Veldhuizen, *Using C++ Template Metaprograms*, C++ Report, Vol 7 No. 4, May 1995, available at `http://extreme.indiana.edu/~tveldhui/papers`) are sometimes too advanced for the template implementation of current compilers.

9.10 Iterator support

9.10.1 Detailed Description

Some compilers still define the iterator struct outside the std namespace, only partially implement it, or do not implement it at all. For the code to be portable these macros should be used instead of the actual functions.

Defines

- #define **QL_ITERATOR** std::iterator
- #define **QL_ITERATOR_TRAITS** std::iterator_traits
- #define **QL_SPECIALIZE_ITERATOR_TRAITS**(T)
- #define **QL_REVERSE_ITERATOR**(iterator, type) std::reverse_iterator< iterator >

  Blame Microsoft for this one...

- #define **QL_FULL_ITERATOR_SUPPORT**

9.10.2 Define Documentation

9.10.2.1 #define QL_ITERATOR std::iterator

Custom iterators should be derived from this struct for the code to be portable.

9.10.2.2 #define QL_ITERATOR_TRAITS std::iterator_traits

For the code to be portable this macro should be used instead of the actual struct.

9.10.2.3 #define QL_SPECIALIZE_ITERATOR_TRAITS(T)

When using the QuantLib implementation of iterator_traits or Visual C++ .Net, this macro might be needed to specialize QL_ITERATOR_TRAITS for a pointer to a user-defined type.

9.10.2.4 #define QL_REVERSE_ITERATOR(iterator, type) std::reverse_iterator< iterator >

Blame Microsoft for this one...

They decided that `std::reverse_iterator<iterator>` needed an extra template argument. For the code to be portable this macro should be used instead of the actual class.

9.10.2.5 #define QL_FULL_ITERATOR_SUPPORT

Some compilers (most notably, Visual C++) still do not fully support iterators in their STL implementation. This macro can be used to select between alternate implementations of blocks of code, namely, one that takes advantage of full iterator support and a less efficient one which is compatible with all compilers.

Chapter 10

QuantLib Namespace Documentation

10.1 QuantLib Namespace Reference

10.1.1 Detailed Description

a.k.a. the QuantLib Foundation

See sect. **Core classes**(p. 27)

Classes

- class **Arguments**
  base class for generic argument groups
- class **Results**
  base class for generic result groups
- class **Array**
  1-D array used in linear algebra.
- class **BlackModel**
  Black-model for vanilla interest-rate derivatives.
- class **CalendarImpl**
  abstract base class for calendar implementations
- class **Calendar**
  calendar class
- class **Calendar::WesternImpl**
- class **CapFlatVolatilityStructure**
  Cap/floor flat volatility structure.

- class **CapletForwardVolatilityStructure**
  Caplet/floorlet forward volatility structure.
- class **CashFlow**
  Base class for cash flows.
- class **IntegerFormatter**
  Formats integers for output.
- class **DoubleFormatter**
  Formats doubles for output.
- class **ArrayFormatter**
  Formats arrays for output.
- class **EuroFormatter**
  Formats amounts in Euro for output.
- class **RateFormatter**
  Formats rates for output.
- class **DateFormatter**
  Formats dates for output.
- class **CurrencyFormatter**
  Formats currencies for output.
- class **StringFormatter**
  Formats strings as lower- or uppercase.
- class **Period**
  Time period described by a number of a given time unit.
- class **Date**
  Concrete date class.
- class **DayCounterImpl**
  abstract base class for day counter implementations
- class **DayCounter**
  day counter class
- class **DiffusionProcess**
  Diffusion process class.
- class **BlackScholesProcess**
  Black-Scholes diffusion process class.
- class **OrnsteinUhlenbeckProcess**
  Ornstein-Uhlenbeck process class.

- class **SquareRootProcess**
  Square-root process class.
- class **DiscretizedAsset**
  Discretized asset class used by numerical methods.
- class **DiscretizedDiscountBond**
  Useful discretized discount bond asset.
- class **DiscretizedOption**
  Discretized option on another asset.
- class **Disposable**
  generic disposable object with move semantics
- class **Error**
  Base error class.
- class **AssertionFailedError**
  Specialized error.
- class **PreconditionNotSatisfiedError**
  Specialized error.
- class **PostconditionNotSatisfiedError**
  Specialized error.
- class **IndexError**
  Specialized error.
- class **IllegalArgumentError**
  Specialized error.
- class **IllegalResultError**
  Specialized error.
- class **OutOfMemoryError**
  Specialized error.
- class **Exercise**
  Exercise(p. 334) *class (American, Bermudan or European).*
- class **AmericanExercise**
  American exercise class.
- class **BermudanExercise**
  Bermudan exercise class.
- class **EuropeanExercise**

European exercise class.

- class **TimeGrid**
  time grid class
- class **Handle**
  Reference-counted pointer.
- class **History**
  Container for historical data.
- class **History::Entry**
  single datum in history
- class **History::const_iterator**
  random access iterator on history entries
- class **Index**
  purely virtual base class for indexes
- class **Instrument**
  Abstract instrument class.
- class **Value**
  pricing results
- struct **Barrier**
  placeholder for enumerated barrier types
- struct **Binary**
  placeholder for enumerated binary types
- class **MarketElement**
  purely virtual base class for market observables
- class **SimpleMarketElement**
  market element returning a stored value
- class **DerivedMarketElement**
  market element whose value depends on another market element
- class **CompositeMarketElement**
  market element whose value depends on two other market element
- class **Null**
  template class providing a null value for a given type.
- class **NumericalMethod**
  Numerical method (Tree, Finite Differences) base class.

- class **Option**
  base option class
- class **Greeks**
  additional option results
- class **Payoff**
  Option(p. 533) *payoff base class.*
- class **StrikedTypePayoff**
- class **PlainVanillaPayoff**
- class **CashOrNothingPayoff**
  Binary(p. 195) *Cash-Or-Nothing option payoff.*
- class **AssetOrNothingPayoff**
  Binary(p. 195) *Asset-Or-Nothing option payoff.*
- class **SupersharePayoff**
  Binary(p. 195) *supershare option payoff.*
- class **PricingEngine**
  base class for pricing engines
- class **Link**
  Relinkable access to a **Handle**(p. 390).
- class **RelinkableHandle**
  Globally accessible relinkable pointer.
- class **RiskStatistics_old**
  Risk analysis tool.
- class **Schedule**
  Payment schedule.
- class **MakeSchedule**
  helper class
- class **Solver1D**
  Base class for 1-D solvers.
- class **SwaptionVolatilityStructure**
  Swaption volatility structure.
- class **TermStructure**
  Term structure.
- class **ZeroYieldStructure**
  Zero yield term structure.

- class **DiscountStructure**
  Discount factor term structure.
- class **ForwardRateStructure**
  Forward rate term structure.
- class **BlackVolTermStructure**
  Black Volatility Term structure.
- class **BlackVolatilityTermStructure**
  Black Volatility term structure.
- class **BlackVarianceTermStructure**
  Black Variance term structure.
- class **LocalVolTermStructure**
  Local Volatility Term structure.

Typedefs

- typedef int **Day**
  Day number.
- typedef int **Year**
  Year number.
- typedef **Schedule Scheduler**
- typedef **MakeSchedule MakeScheduler**
- typedef int **Integer**
  integer number
- typedef double **Real**
  real number
- typedef QL_SIZE_T **Size**
  size of a container
- typedef double **Time**
  continuous quantity with 1-year units
- typedef double **DiscountFactor**
  used to describe discount factors between dates.
- typedef double **Rate**
  used to describe interest rates.
- typedef double **Spread**
  used to describe spreads on interest rates.

Enumerations

- enum **RollingConvention** {
  Preceding, **ModifiedPreceding**, **Following**, **ModifiedFollowing**,
  MonthEndReference }
  Rolling conventions.

- enum **Currency** {
  ARS, **ATS**, **AUD**, **BDT**,
  BEF, **BGL**, **BRL**, **BYB**,
  CAD, **CHF**, **CLP**, **CNY**,
  COP, **CYP**, **CZK**, **DEM**,
  DKK, **EEK**, **EUR**, **GBP**,
  GRD, **HKD**, **HUF**, **ILS**,
  INR, **IQD**, **IRR**, **ISK**,
  ITL, **JPY**, **KRW**, **KWD**,
  LTL, **LVL**, **MTL**, **MXP**,
  NOK, **NPR**, **NZD**, **PKR**,
  PLN, **ROL**, **SAR**, **SEK**,
  SGD, **SIT**, **SKK**, **THB**,
  TRL, **TTD**, **TWD**, **USD**,
  VEB, **ZAR** }
  Known currencies.

- enum **Weekday** {
  Sunday = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,
  Thursday = 5, **Friday** = 6, **Saturday** = 7 }
- enum **Month** {
  January = 1, **February** = 2, **March** = 3, **April** = 4,
  May = 5, **June** = 6, **July** = 7, **August** = 8,
  September = 9, **October** = 10, **November** = 11, **December** = 12 }
  Month names.

- enum **TimeUnit** { **Days** = 0, **Weeks** = 1, **Months** = 2, **Years** = 3 }
  Units used to describe time periods.

Functions

- double **DotProduct** (const **Array** &v1, const **Array** &v2)
- **Disposable**< **Array** > **operator+** (const **Array** &v)
- **Disposable**< **Array** > **operator-** (const **Array** &v)
- **Disposable**< **Array** > **operator+** (const **Array** &v1, const **Array** &v2)
- **Disposable**< **Array** > **operator+** (const **Array** &v1, double a)
- **Disposable**< **Array** > **operator+** (double a, const **Array** &v2)

- **Disposable**< **Array** > **operator-** (const **Array** &v1, const **Array** &v2)
- **Disposable**< **Array** > **operator-** (const **Array** &v1, double a)
- **Disposable**< **Array** > **operator-** (double a, const **Array** &v2)
- **Disposable**< **Array** > **operator** * (const **Array** &v1, const **Array** &v2)
- **Disposable**< **Array** > **operator** * (const **Array** &v1, double a)
- **Disposable**< **Array** > **operator** * (double a, const **Array** &v2)
- **Disposable**< **Array** > **operator/** (const **Array** &v1, const **Array** &v2)
- **Disposable**< **Array** > **operator/** (const **Array** &v1, double a)
- **Disposable**< **Array** > **operator/** (double a, const **Array** &v2)
- **Disposable**< **Array** > **Abs** (const **Array** &v)
- **Disposable**< **Array** > **Sqrt** (const **Array** &v)
- **Disposable**< **Array** > **Log** (const **Array** &v)
- **Disposable**< **Array** > **Exp** (const **Array** &v)
- bool **operator==** (const **Calendar** &c1, const **Calendar** &c2)
- bool **operator!=** (const **Calendar** &c1, const **Calendar** &c2)
- std::ostream & **operator**<< (std::ostream &stream, const **Array** &a)
- std::ostream & **operator**<< (std::ostream &stream, const **Math::Matrix** &matrix)
- std::ostream & **operator**<< (std::ostream &stream, const **Date** &date)
- bool **operator==** (const **Period** &p1, const **Period** &p2)
- bool **operator**< (const **Period** &p1, const **Period** &p2)
- long **operator-** (const **Date** &d1, const **Date** &d2)
- bool **operator==** (const **Date** &d1, const **Date** &d2)
- bool **operator!=** (const **Date** &d1, const **Date** &d2)
- bool **operator**< (const **Date** &d1, const **Date** &d2)
- bool **operator**<= (const **Date** &d1, const **Date** &d2)
- bool **operator**> (const **Date** &d1, const **Date** &d2)
- bool **operator**>= (const **Date** &d1, const **Date** &d2)
- bool **operator==** (const **DayCounter** &d1, const **DayCounter** &d2)
- bool **operator!=** (const **DayCounter** &d1, const **DayCounter** &d2)
- **Disposable**< **Array** > **CenteredGrid** (double center, double dx, **Size** steps)
- **Disposable**< **Array** > **BoundedGrid** (double xMin, double xMax, **Size** steps)

10.1.2 Typedef Documentation

10.1.2.1 typedef Schedule Scheduler

Deprecated
use **Schedule**(p. 588) instead

10.1.2.2 typedef MakeSchedule MakeScheduler

Deprecated
use **MakeSchedule**(p. 469) instead

10.1.3 Enumeration Type Documentation

10.1.3.1 enum RollingConvention

Rolling conventions.

These conventions specify the algorithm used to find the business day which is "closest" to a given holiday.

Enumeration values:

Preceding Choose the first business day before the given holiday.

ModifiedPreceding Choose the first business day before the given holiday unless it belongs to a different month, in which case choose the first business day after the holiday.

Following Choose the first business day after the given holiday.

ModifiedFollowing Choose the first business day after the given holiday unless it belongs to a different month, in which case choose the first business day before the holiday.

MonthEndReference Choose the first business day after the given holiday, if the original date falls on last business day of month result reverts to first business day before month-end

10.1.3.2 enum Currency

Known currencies.

Enumeration values:

ARS Argentinian Peso.

ATS Austrian Schillings.

AUD Australian Dollar.

BDT Bangladesh Taka.

BEF Belgian Franc.

BGL Bulgarian Lev.

BRL Brazilian Real.

BYB Belarusian Ruble.

CAD Canadian Dollar.

CHF Swiss Franc.

CLP Chilean Peso.

CNY Chinese Yuan.

COP Colombian Peso.

CYP Cyprus Pound.

CZK Czech Koruna.

DEM German Mark.

DKK Danish Krone.

EEK Estonian Kroon.

EUR Euro.

GBP British Pound.

GRD Greek Drachma.

HKD Hong Kong Dollar.
HUF Hungarian Forint.
ILS Israeli Shekel.
INR Indian Rupee.
IQD Iraqi Dinar.
IRR Iranian Rial.
ISK Iceland Krona.
ITL Italian Lira.
JPY Japanese Yen.
KRW South-Korean Won.
KWD Kuwaiti dinar.
LTL Lithuanian Litas.
LVL Latvian Lats.
MTL Maltese Lira.
MXP Mexican Peso.
NOK Norwegian Kroner.
NPR Nepal Rupee.
NZD New Zealand Dollar.
PKR Pakistani Rupee.
PLN New Polish Zloty.
ROL Romanian Leu.
SAR Saudi Riyal.
SEK Swedish Krona.
SGD Singapore Dollar.
SIT Slovenian Tolar.
SKK Slovak Koruna.
THB Thai Baht.
TRL Turkish Lira.
TTD Trinidad & Tobago dollar.
TWD Taiwan Dollar.
USD US Dollar.
VEB Venezuelan Bolivar.
ZAR South African Rand.

10.1.3.3 enum Weekday

Day's serial number MOD 7; WEEKDAY Excel function is the same except for Sunday = 7

10.2 QuantLib::Calendars Namespace Reference

10.2.1 Detailed Description

Specialized **Calendar**(p. 235) classes.

See sect. **Calendars**(p. **??**)

Classes

- class **Budapest**
  Budapest calendar
- class **Frankfurt**
  Frankfurt calendar
- class **Helsinki**
  Helsinki calendar
- class **Johannesburg**
  Johannesburg calendar
- class **JointCalendar**
  Joint calendar.
- class **London**
  London calendar
- class **Milan**
  Milan calendar
- class **NewYork**
  New York calendar.
- class **NullCalendar**
  Calendar(p. 235) *for reproducing theoretical calculations.*
- class **Oslo**
  Oslo calendar
- class **Stockholm**
  Stockholm calendar
- class **Sydney**
  Sydney, calendar (New South Wales, Australia)
- class **TARGET**
  TARGET calendar
- class **Tokyo**

Tokyo calendar

- class **Toronto**
  Toronto calendar
- class **Warsaw**
  Warsaw calendar
- class **Wellington**
  Wellington calendar
- class **Zurich**
  Zurich calendar

Enumerations

- enum **JointCalendarRule** { **JoinHolidays**, **JoinBusinessDays** }
  rules for joining calendars

10.2.2 Enumeration Type Documentation

10.2.2.1 enum JointCalendarRule

rules for joining calendars

Enumeration values:

JoinHolidays A date is a holiday for the joint calendar if it is a holiday for any of the given calendars

JoinBusinessDays A date is a business day for the joint calendar if it is a business day for any of the given calendars

10.3 QuantLib::CashFlows Namespace Reference

10.3.1 Detailed Description

Concrete implementations of the **CashFlow**(p. 252) interface.

See sect. cashflows

Classes

- class **BPSCalculator**

  basis point sensitivity (BPS) calculator

- class **Coupon**

  coupon accruing over a fixed period

- class **FixedRateCoupon**

  coupon paying a fixed interest rate

- class **FloatingRateCoupon**

  coupon at par on a term structure

- class **InArrearIndexedCoupon**

  in arrear indexed coupon class

- class **IndexedCoupon**

  base indexed coupon class

- class **ParCoupon**

  coupon at par on a term structure

- class **ShortFloatingRateCoupon**

  short coupon at par on a term structure

- class **Short**

  short indexed coupon

- class **SimpleCashFlow**

  Predetermined cash flow.

- class **TimeBasket**

  Distribution over a number of dates.

- class **UpFrontIndexedCoupon**

  up front indexed coupon class

Functions

- double **BasisPointSensitivity** (const std::vector< **Handle**< **CashFlow** > > &leg, const **RelinkableHandle**< **TermStructure** > &ts)

  Collective basis-point sensitivity of a cash-flow sequence.

- **TimeBasket BasisPointSensitivityBasket** (const std::vector< **Handle**< **CashFlow** > > &leg, const **RelinkableHandle**< **TermStructure** > &ts, int basis)
- std::vector< **Handle**< **CashFlow** > > **FixedRateCouponVector** (const **Schedule** &schedule, const std::vector< double > &nominals, const std::vector< **Rate** > &couponRates, const **DayCounter** &dayCount, const **DayCounter** &firstPeriodDayCount)

  helper function building a sequence of fixed rate coupons

- std::vector< **Handle**< **CashFlow** > > **FixedRateCouponVector** (const std::vector< double > &nominals, const std::vector< **Rate** > &couponRates, const **DayCounter** &dayCount, const **DayCounter** &firstPeriodDayCount, const **Schedule** &schedule)

  helper function building a sequence of fixed rate coupons

- std::vector< **Handle**< **CashFlow** > > **FixedRateCouponVector** (const std::vector< double > &nominals, const std::vector< **Rate** > &couponRates, const **Date** &startDate, const **Date** &endDate, int frequency, const **Calendar** &calendar, **RollingConvention** rollingConvention, bool isAdjusted, const **DayCounter** &dayCount, const **DayCounter** &firstPeriodDayCount, const **Date** &stubDate)

  helper function building a sequence of fixed rate coupons

- std::vector< **Handle**< **CashFlow** > > **FixedRateCouponVector** (const std::vector< double > &nominals, const std::vector< **Rate** > &couponRates, const std::vector< **Date** > &dates, const **Calendar** &calendar, **RollingConvention** roll, const **DayCounter** &dayCounter)

  helper function building a sequence of fixed rate coupons

- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const **Schedule** &schedule, const std::vector< double > &nominals, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads)

  helper function building a sequence of par coupons

- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const **Handle**< **Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **Schedule** &schedule)
- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const **Date** &startDate, const **Date** &endDate, int frequency, const **Calendar** &calendar, **RollingConvention** rollingConvention, const **Handle**< **Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **Date** &stubDate)
- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const std::vector< **Spread** > &spreads, const std::vector< **Date** > &dates, const **Handle**< **Xibor** > &index, int fixingDays, const **Calendar** &calendar, **RollingConvention** roll)
- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const **Date** &startDate, const **Date** &endDate, int frequency, const **Calendar** &calendar, **RollingConvention** rollingConvention, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **Date** &stubDate=**Date**())

  helper function building a sequence of par coupons

- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **Schedule** &schedule)

  helper function building a sequence of par coupons

- std::vector< **Handle**< **CashFlow** > > **FloatingRateCouponVector** (const std::vector< double > &nominals, const std::vector< **Spread** > &spreads, const std::vector< **Date** > &dates, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const **Calendar** &calendar, **RollingConvention** roll)

  helper function building a sequence of par coupons

- template<class IndexedCouponType> std::vector< **Handle**< **CashFlow** > > **IndexedCouponVector** (const **Schedule** &schedule, const std::vector< double > &nominals, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **DayCounter** &dayCounter=**DayCounter**())
- template<class IndexedCouponType> std::vector< **Handle**< **CashFlow** > > **IndexedCouponVector** (const std::vector< double > &nominals, const **Date** &startDate, const **Date** &endDate, int frequency, const **Calendar** &calendar, **RollingConvention** rollingConvention, const **Handle**< **Indexes::Xibor** > &index, int fixingDays, const std::vector< **Spread** > &spreads, const **Date** &stubDate=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

  helper function building a sequence of indexed coupons

10.3.2 Function Documentation

10.3.2.1 std::vector< Handle< CashFlow > > FixedRateCouponVector (const std::vector< double > & *nominals*, const std::vector< Rate > & *couponRates*, const DayCounter & *dayCount*, const DayCounter & *firstPeriodDayCount*, const Schedule & *schedule*)

helper function building a sequence of fixed rate coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.2 std::vector< Handle< CashFlow > > FixedRateCouponVector (const std::vector< double > & *nominals*, const std::vector< Rate > & *couponRates*, const Date & *startDate*, const Date & *endDate*, int *frequency*, const Calendar & *calendar*, RollingConvention *rollingConvention*, bool *isAdjusted*, const DayCounter & *dayCount*, const DayCounter & *firstPeriodDayCount*, const Date & *stubDate* = Date())

helper function building a sequence of fixed rate coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.3 std::vector< Handle< CashFlow > > FixedRateCouponVector (const std::vector< double > & *nominals*, const std::vector< Rate > & *couponRates*, const std::vector< Date > & *dates*, const Calendar & *calendar*, RollingConvention *roll*, const DayCounter & *dayCounter*)

helper function building a sequence of fixed rate coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.4 std::vector< Handle< CashFlow > > FloatingRateCouponVector (const Schedule & *schedule*, const std::vector< double > & *nominals*, const Handle< Indexes::Xibor > & *index*, int *fixingDays*, const std::vector< Spread > & *spreads* = std::vector<Spread>())

helper function building a sequence of par coupons

Warning:
The passing of a non-null stub date - i.e., the creation of a short/long first coupon - is currently disabled.

Todo
A suitable algorithm should be implemented for the calculation of the interpolated index fixing for a short/long first coupon.

10.3.2.5 std::vector<Handle<CashFlow> > FloatingRateCouponVector (const std::vector< double > & *nominals*, const Date & *startDate*, const Date & *endDate*, int *frequency*, const Calendar & *calendar*, RollingConvention *rollingConvention*, const Handle< Indexes::Xibor > & *index*, int *fixingDays*, const std::vector< Spread > & *spreads*, const Date & *stubDate* = Date())

helper function building a sequence of par coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.6 std::vector<Handle<CashFlow> > FloatingRateCouponVector (const std::vector< double > & *nominals*, const Handle< Indexes::Xibor > & *index*, int *fixingDays*, const std::vector< Spread > & *spreads*, const Schedule & *schedule*)

helper function building a sequence of par coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.7 std::vector<Handle<CashFlow> > FloatingRateCouponVector (const std::vector< double > & *nominals*, const std::vector< Spread > & *spreads*, const std::vector< Date > & *dates*, const Handle< Indexes::Xibor > & *index*, int *fixingDays*, const Calendar & *calendar*, RollingConvention *roll*)

helper function building a sequence of par coupons

Deprecated
use the version taking a **Schedule**(p. 588) as first argument instead

10.3.2.8 std::vector<Handle<CashFlow> > IndexedCouponVector (const std::vector< double > & *nominals*, const Date & *startDate*, const Date & *endDate*, int *frequency*, const Calendar & *calendar*, RollingConvention *rollingConvention*, const Handle< Indexes::Xibor > & *index*, int *fixingDays*, const std::vector< Spread > & *spreads*, const Date & *stubDate* = Date(), const DayCounter & *dayCounter* = DayCounter())

helper function building a sequence of indexed coupons

Deprecated
use the version taking a **Schedule**(p. 588) instead

10.4 QuantLib::DayCounters Namespace Reference

10.4.1 Detailed Description

Specialized **DayCounter**(p. 292) classes.

See sect. dayconters

Classes

- class **Actual360**
  Actual/360 day count convention.
- class **Actual365**
  Actual/365 day count convention.
- class **ActualActual**
  Actual/Actual day count.
- class **SimpleDayCounter**
  Simple day counter for reproducing theoretical calculations.
- class **Thirty360**
  30/360 day count convention

10.5 QuantLib::FiniteDifferences Namespace Reference

10.5.1 Detailed Description

Finite difference framework.

See sect. **The finite differences framework**(p. 31)

Classes

- class **AmericanCondition**
- class **BoundaryCondition**

  Abstract boundary condition class for finite difference problems.

- class **NeumannBC**

  Neumann boundary condition (i.e., constant derivative).

- class **DirichletBC**

  Neumann boundary condition (i.e., constant value).

- class **BSMOperator**

  Black-Scholes-Merton differential operator.

- class **CrankNicolson**

  Crank-Nicolson scheme for finite difference methods.

- class **DMinus**

  D_- matricial representation

- class **DPlus**

  D_+ matricial representation

- class **DPlusDMinus**

  D_+D_- matricial representation

- class **DZero**

  D_0 matricial representation

- class **ExplicitEuler**

  Forward Euler scheme for finite difference methods.

- class **FiniteDifferenceModel**

  Generic finite difference model.

- class **ImplicitEuler**

  Backward Euler scheme for finite difference methods.

- class **MixedScheme**

  Mixed (explicit/implicit) scheme for finite difference methods.

- class **OneFactorOperator**

  Interest-rate single factor model differential operator.

- class **ShoutCondition**

  Shout option condition.

- class **StepCondition**

  condition to be applied at every time step

- class **TridiagonalOperator**

  Base implementation for tridiagonal operator.

- class **TridiagonalOperator::TimeSetter**

  encapsulation of time-setting logic

Typedefs

- typedef **FiniteDifferenceModel**< **CrankNicolson**< **TridiagonalOperator** > > **StandardFiniteDifferenceModel**

  default choice for finite-difference model

- typedef **StepCondition**< **Array** > **StandardStepCondition**

  default choice for step condition

Functions

- **Disposable**< **TridiagonalOperator** > **operator+** (const **TridiagonalOperator** &D)
- **Disposable**< **TridiagonalOperator** > **operator-** (const **TridiagonalOperator** &D)
- **Disposable**< **TridiagonalOperator** > **operator+** (const **TridiagonalOperator** &D1, const **TridiagonalOperator** &D2)
- **Disposable**< **TridiagonalOperator** > **operator-** (const **TridiagonalOperator** &D1, const **TridiagonalOperator** &D2)
- **Disposable**< **TridiagonalOperator** > **operator** * (double a, const **TridiagonalOperator** &D)
- **Disposable**< **TridiagonalOperator** > **operator** * (const **TridiagonalOperator** &D, double a)
- **Disposable**< **TridiagonalOperator** > **operator/** (const **TridiagonalOperator** &D, double a)
- double **valueAtCenter** (const **Array** &a)

  mid-point value

- double **firstDerivativeAtCenter** (const **Array** &a, const **Array** &g)

  mid-point first derivative

- double **secondDerivativeAtCenter** (const **Array** &a, const **Array** &g)

  mid-point second derivative

10.6 QuantLib::Indexes Namespace Reference

10.6.1 Detailed Description

Concrete implementations of the **Index**(p. 417) interface.

See sect. indexes

Classes

- class **AUDLibor**
  AUD Libor index (Also known as SIBOR, check settlement days)
- class **CADLibor**
  CAD Libor index (Also known as CDOR)
- class **CHFLibor**
  CHF Libor index (Also known as ZIBOR)
- class **Euribor**
  Euribor index
- class **GBPLibor**
  GBP Libor index
- class **JPYLibor**
  JPY Libor index (Also known as TIBOR, check settlement days)
- class **USDLibor**
  USD Libor index
- class **Xibor**
  base class for libor indexes
- class **XiborManager**
  global repository for libor histories
- class **ZARLibor**
  ZAR Libor index (also known as JIBAR)

10.7 QuantLib::Instruments Namespace Reference

10.7.1 Detailed Description

Concrete implementations of the **Instrument**(p. 422) interface.

See sect. **Instruments and pricers**(p. 45)

Classes

- class **BarrierOption**
  Barrier(p. 180) *option on a single asset.*
- class **BarrierOption::arguments**
  arguments for barrier option calculation
- class **BarrierOption::results**
  results from barrier option calculation
- class **BinaryOption**
  Binary(p. 195) *option on a single asset.*
- class **BinaryOption::arguments**
  arguments for binary option calculation
- class **BinaryOption::results**
  results from binary option calculation
- class **CapFloor**
  Base class for cap-like instruments.
- class **Cap**
  Concrete cap class.
- class **Floor**
  Concrete floor class.
- class **Collar**
  Concrete cap class.
- class **CapFloor::arguments**
  arguments for cap/floor calculation
- class **CapFloor::results**
  results from cap/floor calculation
- class **CliquetOption**
  currently just a container for arguments and results
- class **CliquetOption::arguments**

arguments for cliquet option calculation

- class **ForwardOptionArguments**
  arguments for forward (strike-resetting) option calculation
- class **ForwardVanillaOption**
  Forward version of a vanilla option.
- class **QuantoForwardVanillaOption**
  Quanto version of a forward vanilla option.
- class **QuantoOptionArguments**
  arguments for quanto option calculation
- class **QuantoOptionResults**
  results from quanto option calculation
- class **QuantoVanillaOption**
  quanto version of a vanilla option
- class **SimpleSwap**
  Simple fixed-rate vs Libor swap.
- class **SimpleSwap::arguments**
  arguments for simple swap calculation
- class **SimpleSwap::results**
  results from swaption calculation
- class **Stock**
  Simple stock class.
- class **Swap**
  Interest rate swap.
- class **Swaption**
  Swaption(p. 625) *class.*
- class **Swaption::arguments**
  arguments for swaption calculation
- class **Swaption::results**
  results from swaption calculation
- class **VanillaOption**
  Vanilla option (no discrete dividends, no barriers) on a single asset.
- class **VanillaOption::arguments**
  arguments for vanilla option calculation

- class **VanillaOption::results**
  results from vanilla option calculation

Typedefs

- typedef **CapFloor VanillaCapFloor**
- typedef **Cap VanillaCap**
- typedef **Floor VanillaFloor**
- typedef **Collar VanillaCollar**

10.7.2 Typedef Documentation

10.7.2.1 typedef CapFloor VanillaCapFloor

Deprecated
use **CapFloor**(p. 247) instead

10.7.2.2 typedef Cap VanillaCap

Deprecated
use **Cap**(p. 243) instead

10.7.2.3 typedef Floor VanillaFloor

Deprecated
use **Floor**(p. 357) instead

10.7.2.4 typedef Collar VanillaCollar

Deprecated
use **Collar**(p. 263) instead

10.8 QuantLib::Math Namespace Reference

10.8.1 Detailed Description

Mathematical functions and classes.

See sect. **Math tools**(p. 47)

Classes

- class **BicubicSplineInterpolation**
  bicubic spline interpolation between discrete points
- class **BilinearInterpolation**
  bilinear interpolation between discrete points
- class **CubicSplineInterpolation**
  cubic spline interpolation between discrete points
- class **DiscrepancyStatistics**
  Statistic tool for sequences with discrepancy calculation.
- class **ErrorFunction**
  Error(p. 325) *function.*
- class **GammaFunction**
  Gamma function class.
- class **GaussianStatistics**
  Statistics tool for gaussian-assumption risk measures.
- class **GeneralStatistics**
  Statistics tool.
- class **IncrementalStatistics**
  Statistics tool based on incremental accumulation.
- class **Interpolation**
  abstract base class for 1-D interpolations
- class **Interpolation2D**
  abstract base class for 2-D interpolations
- class **Linear**
  linear interpolation traits
- class **LogLinear**
  log-linear interpolation traits
- class **CubicSpline**

cubic-spline interpolation traits

- class **KronrodIntegral**
  Integral of a 1-dimensional function using the Gauss-Kronrod method.
- class **LexicographicalView**
  Lexicographical 2-D view of a contiguous set of data.
- class **LinearInterpolation**
  linear interpolation between discrete points
- class **LogLinearInterpolation**
  log linear interpolation between discrete points
- class **Matrix**
  matrix used in linear algebra.
- class **MultivariateAccumulator**
  A sample accumulator for multivariate analysis.
- class **NormalDistribution**
  Normal distribution function.
- class **CumulativeNormalDistribution**
  Cumulative normal distribution function.
- class **InverseCumulativeNormal**
  Inverse cumulative normal distribution function.
- class **MoroInverseCumulativeNormal**
  Moro Inverse cumulative normal distribution class.
- class **PrimeNumbers**
  Prime numbers calculator.
- class **RiskMeasures**
- class **GenericRiskStatistics**
  empirical-distribution risk measures
- class **SegmentIntegral**
  Integral of a one-dimensional function.
- class **SequenceStatistics**
  Statistics analysis of N-dimensional (sequence) data.
- class **SimpsonIntegral**
  Integral of a one-dimensional function.
- class **SVD**
  Singular **Value**(p. 662) *Decomposition.*

- class **SymmetricSchurDecomposition**

  symmetric threshold Jacobi algorithm.

- class **TrapezoidIntegral**

  Integral of a one-dimensional function.

Typedefs

- typedef **NormalDistribution GaussianDistribution**
- typedef **InverseCumulativeNormal InvCumulativeNormalDistribution**
- typedef **GaussianStatistics**< **GenericRiskStatistics**< **GeneralStatistics** > > **RiskStatistics**

  default risk measures tool

- typedef **GeneralStatistics Statistics**

  default statistics tool

Enumerations

- enum **SalvagingAlgorithm** { **None**, **Spectral**, **Hypersphere** }

Functions

- template<class F, class R> clipped_function< F, R > **clip** (const F &f, const R &r)
- template<class F, class G> composed_function< F, G > **compose** (const F &f, const G &g)
- **Disposable**< **Matrix** > **pseudoSqrt** (const **Matrix** &realSymmMatrix, SalvagingAlgorithm sa)
- **Disposable**< **Matrix** > **operator+** (const **Matrix** &m1, const **Matrix** &m2)
- **Disposable**< **Matrix** > **operator-** (const **Matrix** &m1, const **Matrix** &m2)
- **Disposable**< **Matrix** > **operator** * (const **Matrix** &m, double x)
- **Disposable**< **Matrix** > **operator** * (double x, const **Matrix** &m)
- **Disposable**< **Matrix** > **operator/** (const **Matrix** &m, double x)
- **Disposable**< **Array** > **operator** * (const **Array** &v, const **Matrix** &m)
- **Disposable**< **Array** > **operator** * (const **Matrix** &m, const **Array** &v)
- **Disposable**< **Matrix** > **operator** * (const **Matrix** &m1, const **Matrix** &m2)
- **Disposable**< **Matrix** > **transpose** (const **Matrix** &m)
- **Disposable**< **Matrix** > **outerProduct** (const **Array** &v1, const **Array** &v2)
- template<class Iterator1, class Iterator2> **Disposable**< **Matrix** > **outerProduct** (Iterator1 v1begin, Iterator1 v1end, Iterator2 v2begin, Iterator2 v2end)
- **Disposable**< **Matrix** > **matrixSqrt** (const **Matrix** &m)
- double **hypot** (const double &a, const double &b)

  Singular **Value**(p. 662) *Decomposition.*

- **Disposable**< **Array** > **SymmetricEigenvalues** (**Matrix** &s)
- **Disposable**< **Matrix** > **SymmetricEigenvectors** (**Matrix** &s)

10.8.2 Function Documentation

10.8.2.1 double hypot (const double & *a*, const double & *b*)

Singular **Value**(p. 662) Decomposition.

Returns:

hypotenuse of real (non-complex) scalars a and b by avoiding underflow/overflow using (a * sqrt(1 + (b/a) * (b/a))), rather than sqrt(a*a + b*b).

10.9 QuantLib::MonteCarlo Namespace Reference

10.9.1 Detailed Description

Monte Carlo framework.

See sect. **The Monte Carlo framework**(p. 36)

Classes

- class **ArithmeticAPOPathPricer_old**
  path pricer for arithmetic average price option
- class **ArithmeticASOPathPricer_old**
  path pricer for arithmetic average strike Asian options
- class **BarrierPathPricer**
  path pricer for **Barrier**(p. 180) *options*
- class **BasketPathPricer_old**
  multipath pricer for European-type basket option
- class **BiasedBarrierPathPricer**
  Biased, but simple, path pricer for **Barrier**(p. 180) *options*
- class **BinaryPathPricer**
  path pricer for **Binary**(p. 195) *options*
- class **BrownianBridge**
  Builds Wiener process paths using Gaussian variates.
- class **CliquetOptionPathPricer**
  path pricer for cliquet options
- class **CliquetOptionPathPricer_old**
  path pricer for cliquet options
- class **EuropeanPathPricer**
  path pricer for European options
- class **EuropeanPathPricer_old**
  path pricer for European options
- class **EverestPathPricer_old**
  path pricer for European-type Everest option
- class **GeometricAPOPathPricer_old**
  path pricer for geometric average price option
- class **GeometricASOPathPricer_old**

path pricer for geometric average strike option

- class **HimalayaPathPricer_old**
  multipath pricer for European-type Himalaya option
- class **MaxBasketPathPricer_old**
  multipath pricer for European-type basket option
- class **MonteCarloModel**
  General purpose Monte Carlo model for path samples.
- class **MultiPath**
  single random walk
- class **MultiPathGenerator**
  Generates a multipath from a random number generator.
- class **MultiPathGenerator_old**
  Generates a multipath from a random number generator.
- class **PagodaPathPricer_old**
  multipath pricer for pagoda options
- class **Path**
  single factor random walk
- class **PathGenerator**
  Generates random paths using a sequence generator.
- class **PathGenerator_old**
  Generates random paths from a random number generator.
- class **PathPricer**
  base class for path pricers
- class **PathPricer_old**
  base class for path pricers
- class **PerformanceOptionPathPricer_old**
  path pricer for performance options
- struct **Sample**
  weighted sample

Typedefs

- typedef GenericPseudoRandom< **RandomNumbers::MersenneTwisterUniformRng**, **Math::InverseCumulativeNormal** > **PseudoRandom**

- typedef **PathGenerator_old**< **RandomNumbers::GaussianRandomGenerator** > **GaussianPathGenerator_old**

  default choice for Gaussian path generator.

- typedef **PathGenerator**< **RandomNumbers::GaussianRandomSequenceGenerator** > **GaussianPathGenerator**
- typedef **MultiPathGenerator_old**< **RandomNumbers::RandomArrayGenerator**< **RandomNumbers::GaussianRandomGenerator** > > **GaussianMultiPathGenerator**

  default choice for Gaussian multi-path generator.

- typedef **MonteCarloModel**< SingleAsset_old< PseudoRandom_old > > **OneFactorMonteCarloOption_old**

  default choice for one-factor Monte Carlo model.

- typedef **MonteCarloModel**< MultiAsset_old< PseudoRandomSequence_old > > **MultiFactorMonteCarloOption**

  default choice for multi-factor Monte Carlo model.

Functions

- template<class DataIterator> **Disposable**< **Math::Matrix** > **getCovariance** (DataIterator volBegin, DataIterator volEnd, const **Math::Matrix** &corr)

10.9.2 Function Documentation

10.9.2.1 Disposable<Math::Matrix> getCovariance (DataIterator *volBegin*, DataIterator *volEnd*, const Math::Matrix & *corr*)

Combines the correlation matrix and the vector of volatilities to return the covariance matrix. Note that only the symmetric part of the correlation matrix is used. Also it is assumed that the diagonal member of the correlation matrix equals one.

10.10 QuantLib::Patterns Namespace Reference

10.10.1 Detailed Description

Implementations of design patterns.

See sect. **Design patterns**(p. 49)

Classes

- class **Bridge**

  The **Bridge**(p. 230) *pattern made explicit.*

- class **CuriouslyRecurringTemplate**

  Support for the curiously recurring template pattern.

- class **LazyObject**

  Framework for calculation on demand and result caching.

- class **Observable**

  Object that notifies its changes to a set of observables.

- class **Observer**

  Object that gets notified when a given observable changes.

- class **AcyclicVisitor**

  degenerate base class for the Acyclic **Visitor**(p. 672) *pattern*

- class **Visitor**

  visitor for a specific class

10.11 QuantLib::Pricers Namespace Reference

10.11.1 Detailed Description

Pricing models for options.

See sect. **Pricers**(p. **??**)

Classes

- class **AnalyticalCapFloor**
  Analytical pricer for cap/floor.
- class **BarrierOption**
  Barrier(p. 180) *option.*
- class **BinaryOption**
  Binary(p. 195) *(digital) option.*
- class **BlackCapFloor**
  CapFloor priced by the Black formula.
- class **BlackSwaption**
  Swaption priced by the Black formula.
- class **CliquetOption**
  cliquet (Ratchet) option
- class **ContinuousGeometricAPO**
  Continuous Geometric Average Price **Option**(p. 533) *(European exercise).*
- class **DiscreteGeometricAPO**
  Discrete Geometric Average Price Asian **Option**(p. 533) *(European style).*
- class **DiscreteGeometricASO**
  Discrete Geometric Average Strike Asian **Option**(p. 533) *(European style).*
- class **EuropeanOption**
  Black-Scholes-Merton European option.
- class **FdAmericanOption**
- class **FdBermudanOption**
  Bermudan option.
- class **FdBsmOption**
  Black-Scholes-Merton option priced numerically.
- class **FdDividendEuropeanOption**
  European option with dividends.

- class **FdDividendShoutOption**
  Shout option with dividends.
- class **FdEuropean**
  Example of European option calculated using finite differences.
- class **FdStepConditionOption**
  option executing additional code at each time step
- class **JamshidianSwaption**
  Jamshidian swaption pricer.
- class **McBasket**
  simple example of multi-factor Monte Carlo pricer
- class **McCliquetOption**
  simple example of Monte Carlo pricer
- class **McDiscreteArithmeticAPO**
  example of Monte Carlo pricer using a control variate
- class **McDiscreteArithmeticASO**
  example of Monte Carlo pricer using a control variate.
- class **McEuropean**
  simple example of Monte Carlo pricer
- class **McEverest**
  Everest-type option pricer.
- class **McHimalaya**
  Himalayan-type option pricer.
- class **McMaxBasket**
  simple example of multi-factor Monte Carlo pricer
- class **McPagoda**
  roofed Asian option
- class **McPerformanceOption**
  Performance option computed using Monte Carlo simulation.
- class **McPricer**
  base class for Monte Carlo pricers
- class **PerformanceOption**
  Performance option.
- class **SingleAssetOption**
  Black-Scholes-Merton option.

- class **TreeCapFloor**

  Cap/Floor priced in a tree.

- class **TreeSwaption**

  Swaption priced on a lattice.

10.12 QuantLib::RandomNumbers Namespace Reference

10.12.1 Detailed Description

Random Number Generators and Low Discrepancy Sequences.

See sect. **Random number and low discrepancy sequence generators**(p. **??**)

Classes

- class **BoxMullerGaussianRng**
  Gaussian random number generator.
- class **CLGaussianRng**
  Gaussian random number generator.
- class **HaltonRsg**
  Halton low-discrepancy sequence generator.
- class **ICGaussianRng**
  Inverse cumulative Gaussian random number generator.
- class **ICGaussianRsg**
  Inverse cumulative Gaussian random sequence generator.
- class **KnuthUniformRng**
  Uniform random number generator.
- class **LecuyerUniformRng**
  Uniform random number generator.
- class **MersenneTwisterUniformRng**
  Uniform random number generator.
- class **RandomArrayGenerator**
  Generates random arrays using a random number generator.
- class **RandomSequenceGenerator**
  Random sequence generator based on a pseudo-random number generator.
- class **SobolRsg**
  Sobol low-discrepancy sequence generator.

Typedefs

- typedef MonteCarlo::PseudoRandom::urng_type **UniformRandomGenerator**
  default choice for uniform random number generator.

- typedef MonteCarlo::PseudoRandom::rng_type **GaussianRandomGenerator**
  default choice for Gaussian random number generator.
- typedef **RandomArrayGenerator**< **GaussianRandomGenerator** > **GaussianArrayGenerator**
  default choice for Gaussian array generator.
- typedef MonteCarlo::PseudoRandom::ursg_type **UniformRandomSequenceGenerator**
  default choice for uniform random sequence generator.
- typedef MonteCarlo::PseudoRandom::rsg_type **GaussianRandomSequenceGenerator**
  default choice for Gaussian random sequence generator.
- typedef **MonteCarlo::LowDiscrepancy::ursg_type UniformLowDiscrepancySequenceGenerator**
- typedef **MonteCarlo::LowDiscrepancy::rsg_type GaussianLowDiscrepancySequenceGenerator**

10.13 QuantLib::ShortRateModels Namespace Reference

10.13.1 Detailed Description

Implementations of short-rate models.

See sect. **The short-rate modelling framework**(p. 42)

Classes

- class **CalibrationHelper**
  liquid market instrument used during calibration
- class **CalibrationSet**
  Set of calibration instruments.
- class **AffineModel**
  Affine model class.
- class **TermStructureConsistentModel**
  Term-structure consistent model class.
- class **Model**
  Abstract short-rate model class.
- class **OneFactorModel**
  Single-factor short-rate model abstract class.
- class **OneFactorModel::ShortRateDynamics**
  Base class describing the short-rate dynamics.
- class **OneFactorModel::ShortRateTree**
  Recombining trinomial tree discretizing the state variable.
- class **OneFactorAffineModel**
  Single-factor affine base class.
- class **BlackKarasinski**
  Standard Black-Karasinski model class.
- class **BlackKarasinski::Dynamics**
  Short-rate dynamics in the Black-Karasinski model.
- class **CoxIngersollRoss**
  Cox-Ingersoll-Ross model class.
- class **CoxIngersollRoss::Dynamics**
  Dynamics(p. 280) *of the short-rate under the Cox-Ingersoll-Ross model.*
- class **ExtendedCoxIngersollRoss**

Extended Cox-Ingersoll-Ross model class.

- class **ExtendedCoxIngersollRoss::Dynamics**

  Short-rate dynamics in the extended Cox-Ingersoll-Ross model.

- class **ExtendedCoxIngersollRoss::FittingParameter**

  Analytical term-structure fitting parameter $\varphi(t)$.

- class **HullWhite**

  Single-factor Hull-White (extended **Vasicek**(p. 669)*) model class.*

- class **HullWhite::Dynamics**

  Short-rate dynamics in the Hull-White model.

- class **HullWhite::FittingParameter**

  Analytical term-structure fitting parameter $\varphi(t)$.

- class **Vasicek**

  Vasicek(p. 669) *model class.*

- class **Vasicek::Dynamics**

  Short-rate dynamics in the **Vasicek**(p. 669) *model.*

- class **ParameterImpl**

  Base class for model parameter implementation.

- class **Parameter**

  Base class for model arguments.

- class **ConstantParameter**

  Standard constant parameter $a(t) = a$.

- class **NullParameter**

  Parameter(p. 539) *which is always zero $a(t) = 0$.*

- class **PiecewiseConstantParameter**

  Piecewise constant parameter.

- class **TermStructureFittingParameter**

  Deterministic time-dependent parameter used for yield-curve fitting.

- class **TwoFactorModel**

  Abstract base-class for two-factor models.

- class **TwoFactorModel::ShortRateDynamics**

  Class describing the dynamics of the two state variables.

- class **TwoFactorModel::ShortRateTree**

  Recombining two-dimensional tree discretizing the state variable.

- class **G2**

  Two-additive-factor gaussian model class.

- class **G2::FittingParameter**

  Analytical term-structure fitting parameter $\varphi(t)$.

10.14 QuantLib::Solvers1D Namespace Reference

10.14.1 Detailed Description

Implementations of the **Solver1D**(p. 608) interface.

See sect. **One-dimensional solvers**(p. **??**)

Classes

- class **Bisection**
  bisection 1-D solver
- class **Brent**
  Brent 1-D solver
- class **FalsePosition**
  False position 1-D solver.
- class **Newton**
  Newton 1-D solver
- class **NewtonSafe**
  safe Newton 1-D solver
- class **Ridder**
  Ridder 1-D solver
- class **Secant**
  secant 1-D solver

10.15 QuantLib::TermStructures Namespace Reference

10.15.1 Detailed Description

Concrete implementations of the **TermStructure**(p. 634) interface.

See sect. **Term structures**(p. 50)

Classes

- class **AffineTermStructure**
  Term-structure implied by an affine model.
- class **DiscountCurve**
  Term structure based on loglinear interpolation of discount factors.
- class **DriftTermStructure**
  Drift term structure.
- class **ExtendedDiscountCurve**
  Term structure based on loglinear interpolation of discount factors.
- class **ForwardSpreadedTermStructure**
  Term structure with added spread on the instantaneous forward rate.
- class **ImpliedTermStructure**
  Implied term structure at a given date in the future.
- class **PiecewiseFlatForward**
  Piecewise flat forward term structure.
- class **QuantoTermStructure**
  Quanto term structure.
- class **RateHelper**
  base class for rate helpers
- class **DepositRateHelper**
  Deposit rate.
- class **FraRateHelper**
  Forward rate agreement.
- class **FuturesRateHelper**
  Interest Rate Futures.
- class **SwapRateHelper**
  Swap rate.
- class **ZeroCurve**

Term structure based on linear interpolation of zero yields.

- class **ZeroSpreadedTermStructure**

  Term structure with an added spread on the zero yield rate.

10.16 QuantLib::Utilities Namespace Reference

10.16.1 Detailed Description

Classes and functions of general utility.

See sect. **Utilities**(p. 51)

Classes

- class **combining_iterator**

  Iterator mapping a function to a set of underlying sequences.

- class **coupling_iterator**

  Iterator mapping a function to a pair of underlying sequences.

- class **filtering_iterator**

  Iterator filtering undesired data.

- struct **lowest_category_iterator**

  most generic of two given iterator categories

- class **processing_iterator**

  Iterator mapping a unary function to an underlying sequence.

- class **stepping_iterator**

  Iterator advancing in constant steps.

Functions

- template<class I, class F> **combining_iterator**< typename 1< I >::value_type, F > **make_combining_iterator** (I it1, I it2, F f)
- template<class It1, class It2, class Function> **coupling_iterator**< It1, It2, Function > **make_coupling_iterator** (It1 it1, It2 it2, Function f)
- template<class Iterator, class UnaryPredicate> **filtering_iterator**< Iterator, UnaryPredicate > **make_filtering_iterator** (Iterator it, UnaryPredicate p, Iterator beforeBegin, Iterator end)
- template<class I, class F> **processing_iterator**< I, F > **make_processing_iterator** (I it, F p)

10.17 QuantLib::VolTermStructures Namespace Reference

10.17.1 Detailed Description

Concrete implementations of Volatility Term Structures.

Classes

- class **BlackConstantVol**
  Constant Black volatility, no time-strike dependence.
- class **BlackVarianceCurve**
  Black volatility curve modelled as variance curve.
- class **BlackVarianceSurface**
  Black volatility surface modelled as variance surface.
- class **ImpliedVolTermStructure**
  Implied vol term structure at a given date in the future.
- class **LocalConstantVol**
  Constant local volatility, no time-strike dependence.
- class **LocalVolCurve**
  Local volatility curve derived from a Black curve.
- class **LocalVolSurface**
  Local volatility surface derived from a Black vol surface.

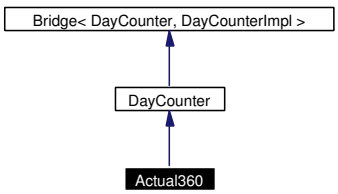
Chapter 11

QuantLib Class Documentation

11.1 Actual360 Class Reference

`#include <actual360.hpp>`

Inheritance diagram for Actual360:



11.1.1 Detailed Description

Actual/360 day count convention.

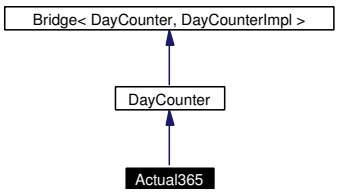
The documentation for this class was generated from the following file:

- ql/DayCounters/**actual360.hpp**

11.2 Actual365 Class Reference

`#include <actual365.hpp>`

Inheritance diagram for Actual365:



11.2.1 Detailed Description

Actual/365 day count convention.

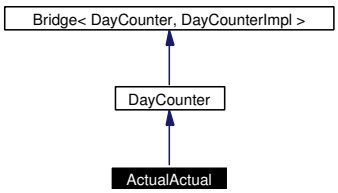
The documentation for this class was generated from the following file:

- ql/DayCounters/**actual365.hpp**

11.3 ActualActual Class Reference

`#include <actualactual.hpp>`

Inheritance diagram for ActualActual:



11.3.1 Detailed Description

Actual/Actual day count.

The day count can be calculated according to ISMA and US Treasury convention, also known as "Actual/Actual (Bond)"; to ISDA, also known as "Actual/Actual (Historical)"; or to AFB, also known as "Actual/Actual (Euro)".

For more details, refer to `http://www.isda.org/c_and_a/pdf/mktc1198.pdf`

Public Types

- enum **Convention** {
  ISMA, **Bond**, **ISDA**, **Historical**,
  AFB, **Euro** }

Public Member Functions

- **ActualActual** (Convention c=ActualActual::ISMA)

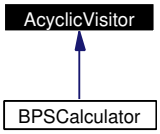
The documentation for this class was generated from the following files:

- ql/DayCounters/**actualactual.hpp**
- ql/DayCounters/**actualactual.cpp**

11.4 AcyclicVisitor Class Reference

`#include <visitor.hpp>`

Inheritance diagram for AcyclicVisitor:



11.4.1 Detailed Description

degenerate base class for the Acyclic **Visitor**(p. 672) pattern

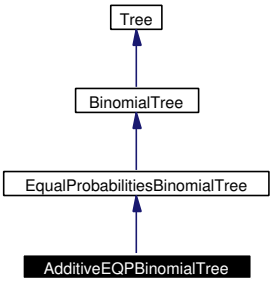
The documentation for this class was generated from the following file:

- ql/Patterns/**visitor.hpp**

11.5 AdditiveEQPBinomialTree Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for AdditiveEQPBinomialTree:



11.5.1 Detailed Description

Additive equal probabilities binomial tree.

Public Member Functions

- **AdditiveEQPBinomialTree** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)

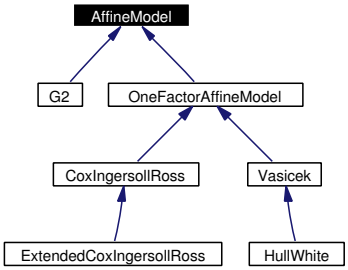
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.6 AffineModel Class Reference

`#include <model.hpp>`

Inheritance diagram for AffineModel:



11.6.1 Detailed Description

Affine model class.

This is the base class for analytically tractable models

Public Member Functions

- virtual **DiscountFactor discount** (**Time** t) const =0

  Implied discount curve.

- virtual double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const =0

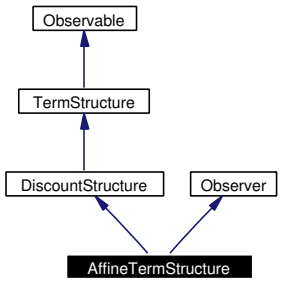
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**model.hpp**

11.7 AffineTermStructure Class Reference

`#include <affinetermstructure.hpp>`

Inheritance diagram for AffineTermStructure:



11.7.1 Detailed Description

Term-structure implied by an affine model.

This class defines a term-structure that is based on an affine model, e.g. Vasicek or Cox-Ingersoll-Ross. It either be instanced using a model with defined arguments, or the model can be calibrated to a set of rate helpers. Of course, there is no point in using a term-structure consistent affine model, since the implied term-structure will just be the initial term-structure on which the model is based.

Public Member Functions

- **AffineTermStructure** (const **Date** &todaysDate, const **Date** &referenceDate, const **Handle**< **ShortRateModels::AffineModel** > &model, const **DayCounter** &dayCounter)

  constructor using a fixed model

- **AffineTermStructure** (const **Date** &todaysDate, const **Date** &referenceDate, const **Handle**< **ShortRateModels::AffineModel** > &model, const std::vector< **Handle**< **RateHelper** > > &instruments, const **Handle**< **Optimization::Method** > &method, const **DayCounter** &dayCounter)

  constructor using a model that has to be calibrated

- **DayCounter dayCounter** () const

  the day counter used for date/time conversion

- **Date todaysDate** () const

  today's date

- **Date referenceDate** () const

  the reference date, i.e., the date at which discount = 1

- **Date maxDate** () const

  the latest date for which the curve can return rates

- void **update** ()

Protected Member Functions

- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const
  discount calculation

11.7.2 Member Function Documentation

11.7.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

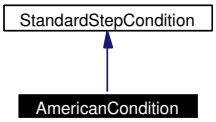
The documentation for this class was generated from the following files:

- ql/TermStructures/**affinetermstructure.hpp**
- ql/TermStructures/**affinetermstructure.cpp**

11.8 AmericanCondition Class Reference

`#include <americancondition.hpp>`

Inheritance diagram for AmericanCondition:



11.8.1 Detailed Description

Todo

Unify the intrinsicValues/Payoff thing

Public Member Functions

- **AmericanCondition** (Option::Type type, double strike)
- **AmericanCondition** (const **Array** &intrinsicValues)
- void **applyTo** (**Array** &a, **Time** t) const
- void **applyTo** (**Handle**< **DiscretizedAsset** > asset) const

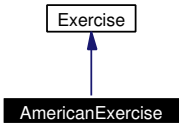
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**americancondition.hpp**

11.9 AmericanExercise Class Reference

`#include <exercise.hpp>`

Inheritance diagram for AmericanExercise:



11.9.1 Detailed Description

American exercise class.

An American option can be exercised at any time between two predefined dates

Public Member Functions

- **AmericanExercise** (**Date** earliestDate, **Date** latestDate)

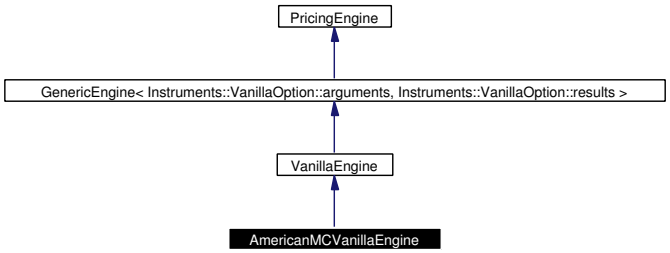
The documentation for this class was generated from the following files:

- ql/**exercise.hpp**
- ql/**exercise.cpp**

11.10 AmericanMCVanillaEngine Class Reference

`#include <americanmcengines.hpp>`

Inheritance diagram for AmericanMCVanillaEngine:



11.10.1 Detailed Description

least-square Monte Carlo engine

Unstable
This engine is known not to work for deeply out-of-the-money options. More problems might surface.

Public Member Functions

- **AmericanMCVanillaEngine** (**Size** requiredSamples, **Size** timeSteps, long seed=0)
- void **calculate** () const

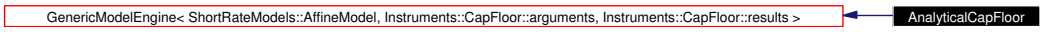
The documentation for this class was generated from the following files:

- ql/PricingEngines/**americanmcengines.hpp**
- ql/PricingEngines/**americanmcengines.cpp**

11.11 AnalyticalCapFloor Class Reference

`#include <analyticalcapfloor.hpp>`

Inheritance diagram for AnalyticalCapFloor:



11.11.1 Detailed Description

Analytical pricer for cap/floor.

Public Member Functions

- **AnalyticalCapFloor** (const **Handle**< **ShortRateModels::AffineModel** > &model)
- void **calculate** () const

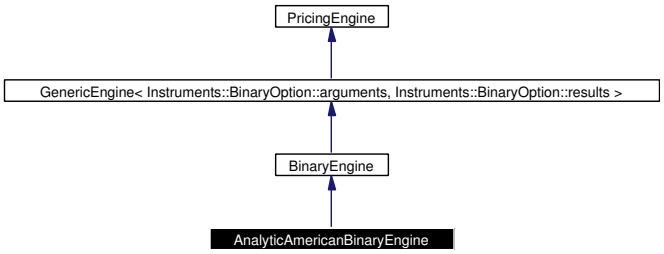
The documentation for this class was generated from the following files:

- ql/Pricers/**analyticalcapfloor.hpp**
- ql/Pricers/**analyticalcapfloor.cpp**

11.12 AnalyticAmericanBinaryEngine Class Reference

`#include <binaryengines.hpp>`

Inheritance diagram for AnalyticAmericanBinaryEngine:



11.12.1 Detailed Description

Pricing engine for American binary options using analytic formulae.

Public Member Functions

- void **calculate** () const

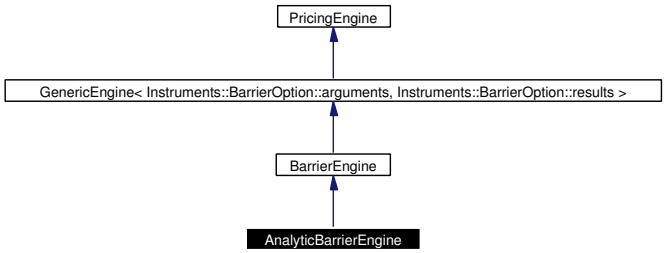
The documentation for this class was generated from the following files:

- ql/PricingEngines/**binaryengines.hpp**
- ql/PricingEngines/**analyticamericanbinaryengine.cpp**

11.13 AnalyticBarrierEngine Class Reference

`#include <barrierengines.hpp>`

Inheritance diagram for AnalyticBarrierEngine:



11.13.1 Detailed Description

Pricing engine for **Barrier**(p. 180) options using analytical formulae.

The formulas are taken from "Option pricing formulas", E.G. Haug, McGraw-Hill, p.69 and following.

Public Member Functions

- void **calculate** () const

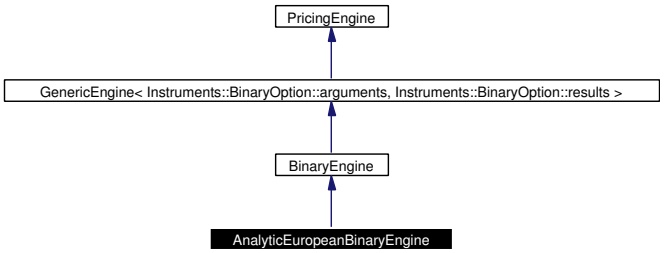
The documentation for this class was generated from the following files:

- ql/PricingEngines/**barrierengines.hpp**
- ql/PricingEngines/analyticbarrierengine.cpp

11.14 AnalyticEuropeanBinaryEngine Class Reference

`#include <binaryengines.hpp>`

Inheritance diagram for AnalyticEuropeanBinaryEngine:



11.14.1 Detailed Description

Pricing engine for European binary options using analytic formulae.

Public Member Functions

- void **calculate** () const

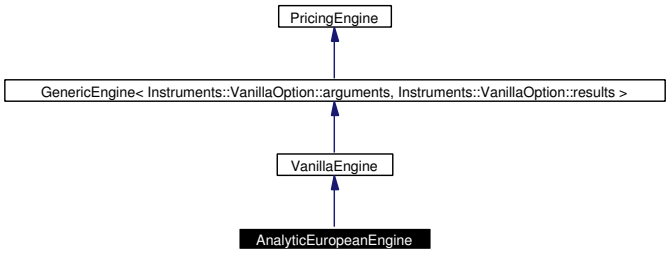
The documentation for this class was generated from the following files:

- ql/PricingEngines/**binaryengines.hpp**
- ql/PricingEngines/**analyticeuropeanbinaryengine.cpp**

11.15 AnalyticEuropeanEngine Class Reference

`#include <vanillaengines.hpp>`

Inheritance diagram for AnalyticEuropeanEngine:



11.15.1 Detailed Description

Pricing engine for European options using analytical formulae.

Public Member Functions

- void **calculate** () const

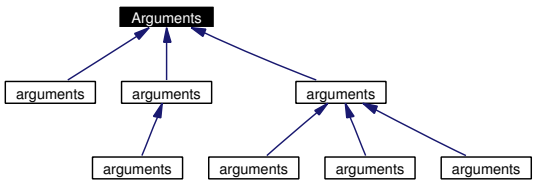
The documentation for this class was generated from the following files:

- ql/PricingEngines/**vanillaengines.hpp**
- ql/PricingEngines/**analyticeuropeanengine.cpp**

11.16 Arguments Class Reference

`#include <argsandresults.hpp>`

Inheritance diagram for Arguments:



11.16.1 Detailed Description

base class for generic argument groups

Public Member Functions

- virtual void **validate** () const =0

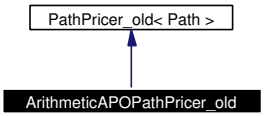
The documentation for this class was generated from the following file:

- ql/**argsandresults.hpp**

11.17 ArithmeticAPOPathPricer_old Class Reference

`#include <arithmeticapopathpricer.hpp>`

Inheritance diagram for ArithmeticAPOPathPricer_old:



11.17.1 Detailed Description

path pricer for arithmetic average price option

Public Member Functions

- **ArithmeticAPOPathPricer_old** (Option::Type type, double underlying, double strike, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

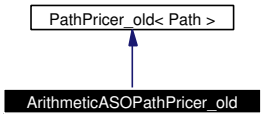
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**arithmeticapopathpricer.hpp**
- ql/MonteCarlo/**arithmeticapopathpricer.cpp**

11.18 ArithmeticASOPathPricer_old Class Reference

`#include <arithmeticasopathpricer.hpp>`

Inheritance diagram for ArithmeticASOPathPricer_old:



11.18.1 Detailed Description

path pricer for arithmetic average strike Asian options

Public Member Functions

- **ArithmeticASOPathPricer_old** (Option::Type type, double underlying, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

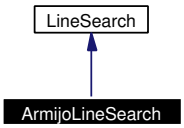
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**arithmeticasopathpricer.hpp**
- ql/MonteCarlo/**arithmeticasopathpricer.cpp**

11.19 ArmijoLineSearch Class Reference

`#include <armijo.hpp>`

Inheritance diagram for ArmijoLineSearch:



11.19.1 Detailed Description

Armijo linesearch.

Let alpha and beta be 2 scalars in [0,1]. Let x be the current value of the unknow, d the search direction and t the step. Let f be the function to minimize. The line search stop when t verifies f(x+t*d) - f(x) <= -alpha*t*f'(x+t*d) and f(x+t/beta*d) - f(x) > -alpha*t*f'(x+t*d)/beta

(see Polak. Algorithms and consitent approximations, Optimization, volume 124 of Applied Mathematical Sciences. Springer-Arrayerlag, N-Y, 1997)

Public Member Functions

- **ArmijoLineSearch** (double eps=1e-8, double alpha=0.5, double beta=0.65)
  Default constructor.
- virtual ~**ArmijoLineSearch** ()
  Destructor.
- virtual double **operator()** (const **Problem** &P, double t_ini)
  Perform line search.

The documentation for this class was generated from the following files:

- ql/Optimization/**armijo.hpp**
- ql/Optimization/**armijo.cpp**

11.20 Array Class Reference

`#include <array.hpp>`

11.20.1 Detailed Description

1-D array used in linear algebra.

This class implements the concept of vector as used in linear algebra. As such, it is **not** meant to be used as a container - `std::vector` should be used instead.

Public Types

- typedef double * **iterator**
- typedef const double * **const_iterator**

Public Member Functions

- typedef **QL_REVERSE_ITERATOR** (iterator, double) reverse_iterator
- typedef **QL_REVERSE_ITERATOR** (const_iterator, double) const _reverse_iterator

Constructors, destructor, and assignment

- **Array** (**Size** size=0)
  creates the array with the given dimension
- **Array** (**Size** size, double value)
  creates the array and fills it with `value`
- **Array** (**Size** size, double value, double increment)
  creates the array and fills it according to $a_0 = value, a_i = a_{i-1} + increment$
- **Array** (const **Array** &)
- **Array** (const **Disposable**< **Array** > &)
- **Array** & **operator=** (const **Array** &)
- **Array** & **operator=** (const **Disposable**< **Array** > &)

Vector algebra

`v` += `x` *and similar operation involving a scalar value are shortcuts for* $\forall i : v_i = v_i + x$

`v` *== `w` *and similar operation involving two vectors are shortcuts for* $\forall i : v_i = v_i \times w_i$

Precondition:
all arrays involved in an algebraic expression must have the same size.

- **Array** & **operator+=** (const **Array** &)
- **Array** & **operator+=** (double)
- **Array** & **operator-=** (const **Array** &)
- **Array** & **operator-=** (double)
- **Array** & **operator** *= (const **Array** &)
- **Array** & **operator** *= (double)
- **Array** & **operator/=** (const **Array** &)

- **Array** & **operator/=** (double)

Element access

- double **operator[ ]** (**Size**) const
  read-only
- double & **operator[ ]** (**Size**)
  read-write

Inspectors

- **Size size** () const
  dimension of the array

Iterator access

- const_iterator **begin** () const
- iterator **begin** ()
- const_iterator **end** () const
- iterator **end** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rend** () const
- reverse_iterator **rend** ()

Utilities

- void **swap** (**Array** &)

Related Functions

(Note that these are not member functions.)

- double **DotProduct** (const **Array** &, const **Array** &)
- **Disposable**< **Array** > **operator+** (const **Array** &v)
- **Disposable**< **Array** > **operator-** (const **Array** &v)
- **Disposable**< **Array** > **operator+** (const **Array** &, const **Array** &)
- **Disposable**< **Array** > **operator+** (const **Array** &, double)
- **Disposable**< **Array** > **operator+** (double, const **Array** &)
- **Disposable**< **Array** > **operator-** (const **Array** &, const **Array** &)
- **Disposable**< **Array** > **operator-** (const **Array** &, double)
- **Disposable**< **Array** > **operator-** (double, const **Array** &)
- **Disposable**< **Array** > **operator** * (const **Array** &, const **Array** &)
- **Disposable**< **Array** > **operator** * (const **Array** &, double)
- **Disposable**< **Array** > **operator** * (double, const **Array** &)
- **Disposable**< **Array** > **operator/** (const **Array** &, const **Array** &)
- **Disposable**< **Array** > **operator/** (const **Array** &, double)
- **Disposable**< **Array** > **operator/** (double, const **Array** &)
- **Disposable**< **Array** > **Abs** (const **Array** &)

- **Disposable**< **Array** > **Sqrt** (const **Array** &)
- **Disposable**< **Array** > **Log** (const **Array** &)
- **Disposable**< **Array** > **Exp** (const **Array** &)
- std::ostream & **operator**<< (std::ostream &, const **Array** &)

The documentation for this class was generated from the following files:

- ql/**array.hpp**
- ql/**dataformatters.hpp**

11.21 ArrayFormatter Class Reference

`#include <dataformatters.hpp>`

11.21.1 Detailed Description

Formats arrays for output.

Static Public Member Functions

- template<class DataIterator> std::string **toString** (DataIterator begin, DataIterator end, int precision=6, int digits=0, **Size** elementsPerRow=QL_MAX_INT)

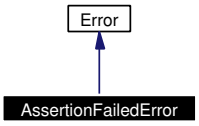
The documentation for this class was generated from the following file:

- ql/**dataformatters.hpp**

11.22 AssertionFailedError Class Reference

`#include <errors.hpp>`

Inheritance diagram for AssertionFailedError:



11.22.1 Detailed Description

Specialized error.

Thrown upon a failed assertion.

Public Member Functions

- **AssertionFailedError** (const std::string &what="")

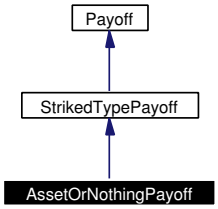
The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.23 AssetOrNothingPayoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for AssetOrNothingPayoff:



11.23.1 Detailed Description

Binary(p. 195) Asset-Or-Nothing option payoff.

Public Member Functions

- **AssetOrNothingPayoff** (Option::Type type, double strike)
- double **operator()** (double price) const

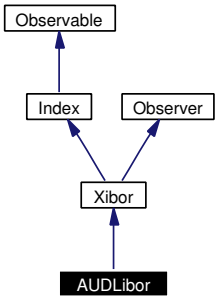
The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.24 AUDLibor Class Reference

`#include <audlibor.hpp>`

Inheritance diagram for AUDLibor:



11.24.1 Detailed Description

AUD Libor index (Also known as SIBOR, check settlement days)

Public Member Functions

- **AUDLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual365**())

The documentation for this class was generated from the following file:

- ql/Indexes/**audlibor.hpp**

11.25 Barrier Struct Reference

`#include <barrieroption.hpp>`

11.25.1 Detailed Description

placeholder for enumerated barrier types

Public Types

- enum **Type** { **DownIn**, **UpIn**, **DownOut**, **UpOut** }

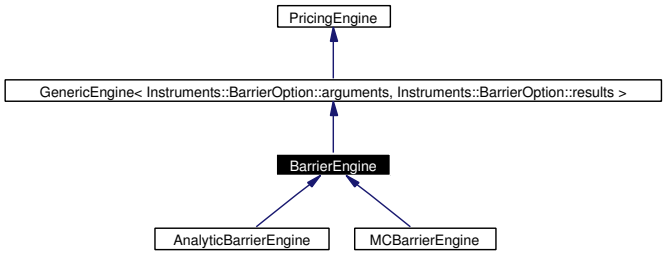
The documentation for this struct was generated from the following file:

- ql/Instruments/**barrieroption.hpp**

11.26 BarrierEngine Class Reference

`#include <barrierengines.hpp>`

Inheritance diagram for BarrierEngine:



11.26.1 Detailed Description

Barrier(p. 180) engine base class.

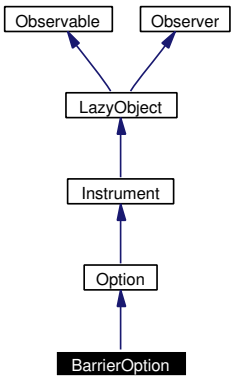
The documentation for this class was generated from the following file:

- ql/PricingEngines/**barrierengines.hpp**

11.27 BarrierOption Class Reference

`#include <barrieroption.hpp>`

Inheritance diagram for BarrierOption:



11.27.1 Detailed Description

Barrier(p. 180) option on a single asset.

Public Member Functions

- **BarrierOption** (Barrier::Type barrierType, double barrier, double rebate, Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, double strike, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >(), const std::string &isinCode="", const std::string &description="")
- bool **isExpired** () const

  returns whether the instrument is still tradable.

- void **setupArguments** (**Arguments** *) const

greeks

- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- double **strikeSensitivity** () const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- double **delta_**
- double **gamma_**
- double **theta_**
- double **vega_**
- double **rho_**
- double **dividendRho_**
- double **strikeSensitivity_**
- Barrier::Type **barrierType_**
- double **barrier_**
- double **rebate_**
- Option::Type **type_**
- **RelinkableHandle< MarketElement > underlying_**
- double **strike_**
- **Exercise exercise_**
- **RelinkableHandle< TermStructure > riskFreeTS_**
- **RelinkableHandle< TermStructure > dividendTS_**
- **RelinkableHandle< BlackVolTermStructure > volTS_**

11.27.2 Member Function Documentation

11.27.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

11.27.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from **Instrument** (p. 424).

11.27.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

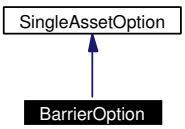
Reimplemented from **Instrument** (p. 424).

The documentation for this class was generated from the following files:

- ql/Instruments/**barrieroption.hpp**
- ql/Instruments/**barrieroption.cpp**

11.28 BarrierOption Class Reference

Inheritance diagram for BarrierOption:



11.28.1 Detailed Description

Barrier(p. 180) option.

The analytical calculation are taken from "Option pricing formulas", E.G. Haug, McGraw-Hill, p.69 and following.

Deprecated
use **Instruments::BarrierOption**(p. 182) instead.

Public Member Functions

- **BarrierOption** (Barrier::Type barrType, Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, double barrier, double rebate=0.0)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- **Handle**< **SingleAssetOption** > **clone** () const

Protected Member Functions

- void **calculate_** () const

Protected Attributes

- double **greeksCalculated_**
- double **delta_**
- double **gamma_**
- double **theta_**

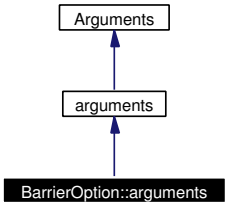
The documentation for this class was generated from the following files:

- ql/Pricers/**barrieroptionpricer.hpp**
- ql/Pricers/**barrieroptionpricer.cpp**

11.29 BarrierOption::arguments Class Reference

`#include <barrieroption.hpp>`

Inheritance diagram for BarrierOption::arguments:



11.29.1 Detailed Description

arguments for barrier option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Barrier::Type **barrierType**
- double **barrier**
- double **rebate**

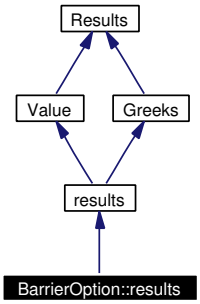
The documentation for this class was generated from the following file:

- ql/Instruments/**barrieroption.hpp**

11.30 BarrierOption::results Class Reference

`#include <barrieroption.hpp>`

Inheritance diagram for BarrierOption::results:



11.30.1 Detailed Description

results from barrier option calculation

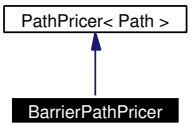
The documentation for this class was generated from the following file:

- ql/Instruments/**barrieroption.hpp**

11.31 BarrierPathPricer Class Reference

`#include <barrierpathpricer.hpp>`

Inheritance diagram for BarrierPathPricer:



11.31.1 Detailed Description

path pricer for **Barrier**(p. 180) options

Public Member Functions

- **BarrierPathPricer** (Barrier::Type barrierType, double barrier, double rebate, Option::Type type, double underlying, double strike, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Handle**< **DiffusionProcess** > &diffProcess, **RandomNumbers::UniformRandomSequenceGenerator** sequenceGen)
- double **operator()** (const **Path** &path) const

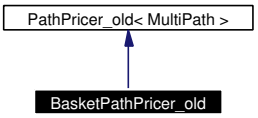
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**barrierpathpricer.hpp**
- ql/MonteCarlo/**barrierpathpricer.cpp**

11.32 BasketPathPricer_old Class Reference

`#include <basketpathpricer.hpp>`

Inheritance diagram for BasketPathPricer_old:



11.32.1 Detailed Description

multipath pricer for European-type basket option

The value of the option at expiration is given by the value of the underlying which has best performed.

Public Member Functions

- **BasketPathPricer_old** (Option::Type type, const std::vector< double > &underlying, double strike, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **MultiPath** &multiPath) const

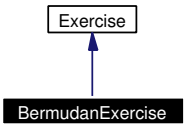
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**basketpathpricer.hpp**
- ql/MonteCarlo/**basketpathpricer.cpp**

11.33 BermudanExercise Class Reference

`#include <exercise.hpp>`

Inheritance diagram for BermudanExercise:



11.33.1 Detailed Description

Bermudan exercise class.

A Bermudan option can only be exercised at a set of fixed dates.

Public Member Functions

- **BermudanExercise** (const std::vector< **Date** > &dates)

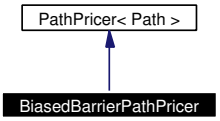
The documentation for this class was generated from the following files:

- ql/**exercise.hpp**
- ql/**exercise.cpp**

11.34 BiasedBarrierPathPricer Class Reference

`#include <biasedbarrierpathpricer.hpp>`

Inheritance diagram for BiasedBarrierPathPricer:



11.34.1 Detailed Description

Biased, but simple, path pricer for **Barrier**(p. 180) options

Public Member Functions

- **BiasedBarrierPathPricer** (Barrier::Type barrierType, double barrier, double rebate, Option::Type type, double underlying, double strike, const **RelinkableHandle**< **TermStructure** > &riskFreeTS)
- double **operator()** (const **Path** &path) const

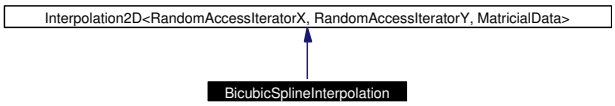
The documentation for this class was generated from the following files:

- ql/MonteCarlo/biasedbarrierpathpricer.hpp
- ql/MonteCarlo/biasedbarrierpathpricer.cpp

11.35 BicubicSplineInterpolation Class Template Reference

`#include <bicubicsplineinterpolation.hpp>`

Inheritance diagram for BicubicSplineInterpolation:



11.35.1 Detailed Description

template<class RandomAccessIteratorX, class RandomAccessIteratorY, class MatricialData> class QuantLib::Math::BicubicSplineInterpolation< RandomAccessIteratorX, RandomAccessIteratorY, MatricialData >

bicubic spline interpolation between discrete points

Public Types

- typedef double **result_type**

Public Member Functions

- **BicubicSplineInterpolation** (const RandomAccessIteratorX &xBegin, const RandomAccessIteratorX &xEnd, const RandomAccessIteratorY &yBegin, const RandomAccessIteratorY &yEnd, const MatricialData &data)
- result_type **operator()** (const first_argument_type &x, const second_argument_type &y, bool allowExtrapolation=false) const

Public Attributes

- typedef< RandomAccessIteratorX >::value_type **first_argument_type**
- typedef< RandomAccessIteratorY >::value_type **second_argument_type**

11.35.2 Member Function Documentation

11.35.2.1 result_type operator() (const first_argument_type & *x*, const second_argument_type & *y*, bool *allowExtrapolation* = false) const [virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implements **Interpolation2D** (p. 430).

The documentation for this class was generated from the following file:

- ql/Math/**bicubicsplineinterpolation.hpp**

11.36 BilinearInterpolation Class Template Reference

`#include <bilinearinterpolation.hpp>`

Inheritance diagram for BilinearInterpolation:

11.36.1 Detailed Description

template<class RandomAccessIteratorX, class RandomAccessIteratorY, class MatricialData> class QuantLib::Math::BilinearInterpolation< RandomAccessIteratorX, RandomAccessIteratorY, MatricialData >

bilinear interpolation between discrete points

Public Types

- typedef double **result_type**

Public Member Functions

- **BilinearInterpolation** (const RandomAccessIteratorX &xBegin, const RandomAccessIteratorX &xEnd, const RandomAccessIteratorY &yBegin, const RandomAccessIteratorY &yEnd, const MatricialData &data)
- result_type **operator()** (const first_argument_type &x, const second_argument_type &y, bool allowExtrapolation=false) const

Public Attributes

- typedef< RandomAccessIteratorX >::value_type **first_argument_type**
- typedef< RandomAccessIteratorY >::value_type **second_argument_type**

11.36.2 Member Function Documentation

11.36.2.1 result_type operator() (const first_argument_type & *x*, const second_argument_type & *y*, bool *allowExtrapolation* = false) const [virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implements **Interpolation2D** (p. 430).

The documentation for this class was generated from the following file:

- ql/Math/**bilinearinterpolation.hpp**

11.37 Binary Struct Reference

`#include <binaryoption.hpp>`

11.37.1 Detailed Description

placeholder for enumerated binary types

Public Types

- enum **Type** { **CashAtHit**, **CashAtExpiry** }

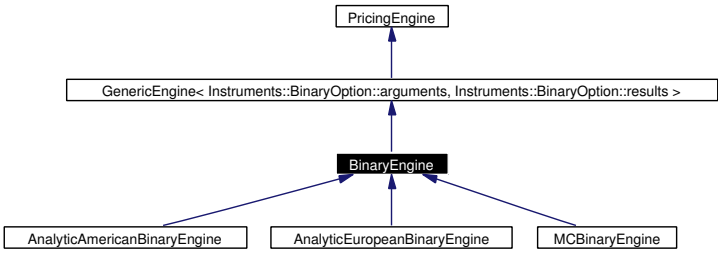
The documentation for this struct was generated from the following file:

- ql/Instruments/**binaryoption.hpp**

11.38 BinaryEngine Class Reference

`#include <binaryengines.hpp>`

Inheritance diagram for BinaryEngine:



11.38.1 Detailed Description

Binary(p. 195) engine base class.

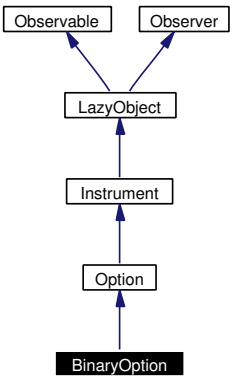
The documentation for this class was generated from the following file:

- ql/PricingEngines/**binaryengines.hpp**

11.39 BinaryOption Class Reference

`#include <binaryoption.hpp>`

Inheritance diagram for BinaryOption:



11.39.1 Detailed Description

Binary(p. 195) option on a single asset.

Public Member Functions

- **BinaryOption** (Binary::Type binaryType, double barrier, double cashPayoff, Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >(), const std::string &isinCode="", const std::string &description="")
- bool **isExpired** () const

  returns whether the instrument is still tradable.

- void **setupArguments** (**Arguments** *) const

greeks

- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- double **strikeSensitivity** () const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- double **delta_**
- double **gamma_**
- double **theta_**
- double **vega_**
- double **rho_**
- double **dividendRho_**
- double **strikeSensitivity_**
- Binary::Type **binaryType_**
- double **barrier_**
- double **cashPayoff_**
- Option::Type **type_**
- **RelinkableHandle< MarketElement > underlying_**
- **Exercise exercise_**
- **RelinkableHandle< TermStructure > riskFreeTS_**
- **RelinkableHandle< TermStructure > dividendTS_**
- **RelinkableHandle< BlackVolTermStructure > volTS_**

11.39.2 Member Function Documentation

11.39.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

11.39.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from **Instrument** (p. 424).

11.39.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. 424).

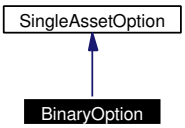
The documentation for this class was generated from the following files:

- ql/Instruments/**binaryoption.hpp**
- ql/Instruments/**binaryoption.cpp**

11.40 BinaryOption Class Reference

`#include <binaryoptionpricer.hpp>`

Inheritance diagram for BinaryOption:



11.40.1 Detailed Description

Binary(p. 195) (digital) option.

Deprecated
use **Instruments::BarrierOption**(p. 182) instead.

Public Member Functions

- **BinaryOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, double cashPayoff)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- **Handle**< **SingleAssetOption** > **clone** () const

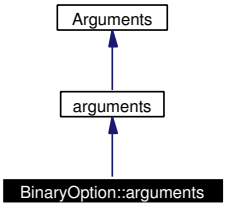
The documentation for this class was generated from the following files:

- ql/Pricers/**binaryoptionpricer.hpp**
- ql/Pricers/**binaryoptionpricer.cpp**

11.41 BinaryOption::arguments Class Reference

`#include <binaryoption.hpp>`

Inheritance diagram for BinaryOption::arguments:



11.41.1 Detailed Description

arguments for binary option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- Binary::Type **binaryType**
- double **barrier**
- double **cashPayoff**

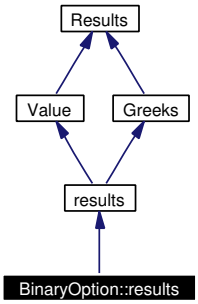
The documentation for this class was generated from the following files:

- ql/Instruments/**binaryoption.hpp**
- ql/Instruments/**binaryoption.cpp**

11.42 BinaryOption::results Class Reference

`#include <binaryoption.hpp>`

Inheritance diagram for BinaryOption::results:



11.42.1 Detailed Description

results from binary option calculation

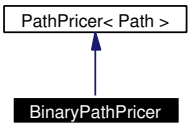
The documentation for this class was generated from the following file:

- ql/Instruments/**binaryoption.hpp**

11.43 BinaryPathPricer Class Reference

`#include <binarypathpricer.hpp>`

Inheritance diagram for BinaryPathPricer:



11.43.1 Detailed Description

path pricer for **Binary**(p. 195) options

Public Member Functions

- **BinaryPathPricer** (Binary::Type binaryType, double barrier, double cashPayoff, Option::Type type, double underlying, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Handle**< **DiffusionProcess** > &diffProcess, **RandomNumbers::UniformRandomSequenceGenerator** sequenceGen)
- double **operator()** (const **Path** &path) const

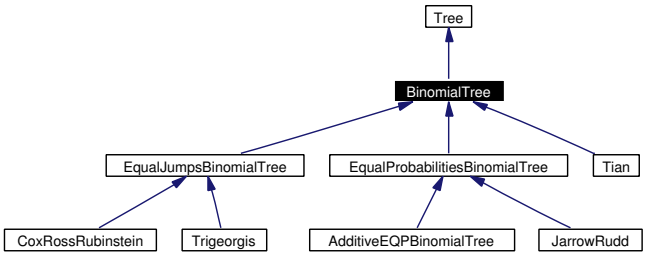
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**binarypathpricer.hpp**
- ql/MonteCarlo/**binarypathpricer.cpp**

11.44 BinomialTree Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for BinomialTree:



11.44.1 Detailed Description

Binomial tree base class.

Public Member Functions

- **BinomialTree** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)
- **Size size** (**Size** i) const
- **Size descendant** (**Size** i, **Size** index, **Size** branch) const
- virtual double **underlying** (**Size** i, **Size** index) const =0
- virtual double **probability** (**Size** i, **Size** index, **Size** branch) const =0

Protected Attributes

- double **x0_**
- double **driftPerStep_**
- **Time dt_**

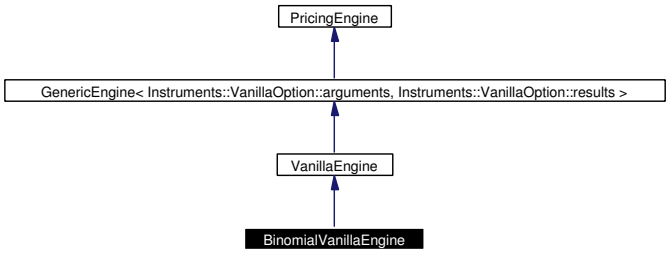
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.45 BinomialVanillaEngine Class Reference

`#include <vanillaengines.hpp>`

Inheritance diagram for BinomialVanillaEngine:



11.45.1 Detailed Description

Pricing engine for Vanilla options using binomial trees.

Public Types

- enum **Type** {
  CoxRossRubinstein, **JarrowRudd**, **EQP**, **Trigeorgis**,
  Tian }

Public Member Functions

- **BinomialVanillaEngine** (Type type, **Size** timeSteps)
- void **calculate** () const

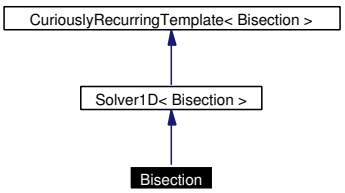
The documentation for this class was generated from the following files:

- ql/PricingEngines/**vanillaengines.hpp**
- ql/PricingEngines/**binomialvanillaengine.cpp**

11.46 Bisection Class Reference

`#include <bisection.hpp>`

Inheritance diagram for Bisection:



11.46.1 Detailed Description

bisection 1-D solver

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

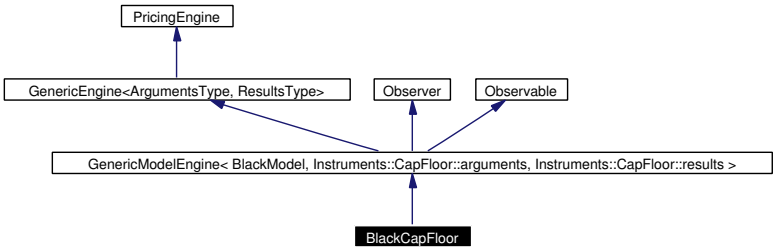
The documentation for this class was generated from the following file:

- ql/Solvers1D/**bisection.hpp**

11.47 BlackCapFloor Class Reference

`#include <blackcapfloor.hpp>`

Inheritance diagram for BlackCapFloor:



11.47.1 Detailed Description

CapFloor priced by the Black formula.

Public Member Functions

- **BlackCapFloor** (const **Handle**< **BlackModel** > &mod)
- void **calculate** () const

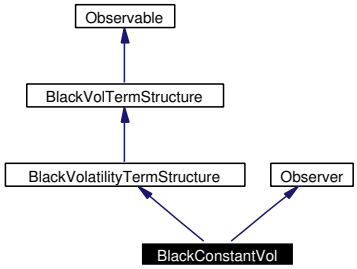
The documentation for this class was generated from the following files:

- ql/Pricers/**blackcapfloor.hpp**
- ql/Pricers/**blackcapfloor.cpp**

11.48 BlackConstantVol Class Reference

`#include <blackconstantvol.hpp>`

Inheritance diagram for BlackConstantVol:



11.48.1 Detailed Description

Constant Black volatility, no time-strike dependence.

This class implements the **BlackVolatilityTermStructure**(p. 221) interface for a constant Black volatility (no time/strike dependence).

Public Member Functions

- **BlackConstantVol** (const **Date** &referenceDate, double volatility, const **DayCounter** &dayCounter=**DayCounters::Actual365**())
- **BlackConstantVol** (const **Date** &referenceDate, const **RelinkableHandle**< **MarketElement** > &volatility, const **DayCounter** &dayCounter=**DayCounters::Actual365**())

BlackVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols
- double **blackForwardVol** (**Time** t1, **Time** t2, double strike, bool extrapolate=false) const
  future (a.k.a. forward) volatility

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **blackVolImpl** (**Time** t, double, bool extrapolate=false) const

  implements the actual Black vol calculation in derived classes

11.48.2 Member Function Documentation

11.48.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

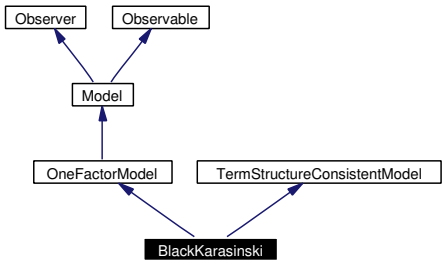
The documentation for this class was generated from the following file:

- ql/Volatilities/**blackconstantvol.hpp**

11.49 BlackKarasinski Class Reference

`#include <blackkarasinski.hpp>`

Inheritance diagram for BlackKarasinski:



11.49.1 Detailed Description

Standard Black-Karasinski model class.

This class implements the standard Black-Karasinski model defined by

$$d \ln r_t = (\theta(t) - \alpha \ln r_t)dt + \sigma dW_t,$$

where *alpha* and *sigma* are constants.

Public Member Functions

- **BlackKarasinski** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.1)
- **Handle**< ShortRateDynamics > **dynamics** () const

  returns the short-rate dynamics

- **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Return by default a trinomial recombining tree.

The documentation for this class was generated from the following files:

- ql/ShortRateModels/OneFactorModels/**blackkarasinski.hpp**
- ql/ShortRateModels/OneFactorModels/**blackkarasinski.cpp**

11.50 BlackKarasinski::Dynamics Class Reference

`#include <blackkarasinski.hpp>`

11.50.1 Detailed Description

Short-rate dynamics in the Black-Karasinski model.

The short-rate is here

$$r_t = e^{\varphi(t)+x_t}$$

where $\varphi(t)$ is the deterministic time-dependent parameter (which can not be determined analytically) used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

Public Member Functions

- **Dynamics** (const **Parameter** &fitting, double alpha, double sigma)
- double **variable** (**Time** t, **Rate** r) const
- double **shortRate** (**Time** t, double x) const

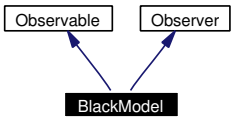
The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**blackkarasinski.hpp**

11.51 BlackModel Class Reference

`#include <blackmodel.hpp>`

Inheritance diagram for BlackModel:



11.51.1 Detailed Description

Black-model for vanilla interest-rate derivatives.

Public Member Functions

- **BlackModel** (const **RelinkableHandle**< **MarketElement** > &volatility, const **RelinkableHandle**< **TermStructure** > &termStructure)
- void **update** ()
- double **volatility** () const
- const **RelinkableHandle**< **TermStructure** > & **termStructure** () const

Static Public Member Functions

- double **formula** (double f, double k, double v, double w)

  Black formula.

11.51.2 Member Function Documentation

11.51.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.51.2.2 double formula (double *f*, double *k*, double *v*, double *w*) [static]

Black formula.

Returns

$$Black(f,k,v,w) = fw\Phi(wd_1(f,k,v)) - kw\Phi(wd_2(f,k,v)),$$

where

$$d_1(f,k,v) = \frac{\ln(f/k) + v^2/2}{v}$$

and

$$d_2(f,k,v) = d_1(f,k,v) - v.$$

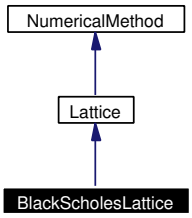
The documentation for this class was generated from the following file:

- ql/**blackmodel.hpp**

11.52 BlackScholesLattice Class Reference

`#include <bsmlattice.hpp>`

Inheritance diagram for BlackScholesLattice:



11.52.1 Detailed Description

Simple binomial lattice approximating the Black-Scholes model.

Public Member Functions

- **BlackScholesLattice** (const **Handle**< **Tree** > &tree, **Rate** riskFreeRate, **Time** end, **Size** steps)
- **Size size** (**Size** i) const
- **DiscountFactor discount** (**Size** i, **Size** j) const

  Discount factor at time t_i and node indexed by index.

- const **Handle**< **Tree** > & **tree** () const

Protected Member Functions

- void **stepback** (**Size** i, const **Array** &values, **Array** &newValues) const
- **Size descendant** (**Size** i, **Size** index, **Size** branch) const

  Tree(p. 648) *properties.*

- double **probability** (**Size** i, **Size** index, **Size** branch) const

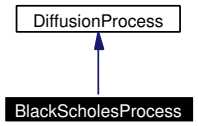
The documentation for this class was generated from the following files:

- ql/Lattices/**bsmlattice.hpp**
- ql/Lattices/**bsmlattice.cpp**

11.53 BlackScholesProcess Class Reference

`#include <diffusionprocess.hpp>`

Inheritance diagram for BlackScholesProcess:



11.53.1 Detailed Description

Black-Scholes diffusion process class.

This class describes the stochastic process governed by

$$dS(t,S) = (r(t) - q(t) - \frac{\sigma(t,S)^2}{2})dt + \sigma dW_t.$$

Public Member Functions

- **BlackScholesProcess** (const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **BlackVolTermStructure** > &blackVolTS, double s0)
- double **drift** (**Time** t, double x) const

  returns the drift part of the equation, i.e. $\mu(t,x_t)$

- double **diffusion** (**Time** t, double x) const

  returns the diffusion part of the equation, i.e. $\sigma(t,x_t)$

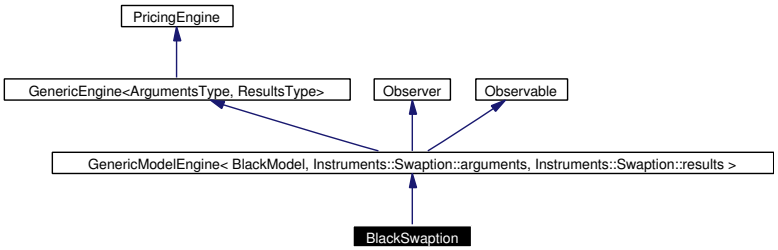
The documentation for this class was generated from the following files:

- ql/**diffusionprocess.hpp**
- ql/**diffusionprocess.cpp**

11.54 BlackSwaption Class Reference

`#include <blackswaption.hpp>`

Inheritance diagram for BlackSwaption:



11.54.1 Detailed Description

Swaption priced by the Black formula.

Public Member Functions

- **BlackSwaption** (const **Handle**< **BlackModel** > &mod)
- void **calculate** () const

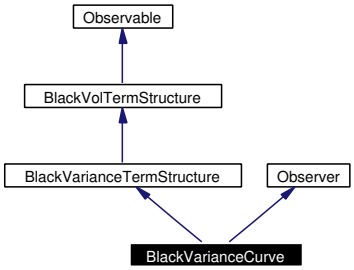
The documentation for this class was generated from the following files:

- ql/Pricers/**blackswaption.hpp**
- ql/Pricers/**blackswaption.cpp**

11.55 BlackVarianceCurve Class Reference

`#include <blackvariancecurve.hpp>`

Inheritance diagram for BlackVarianceCurve:



11.55.1 Detailed Description

Black volatility curve modelled as variance curve.

This class calculates time-dependent Black volatilities using as input a vector of (ATM) Black volatilities observed in the market.

The calculation is performed interpolating on the variance curve. Linear interpolation is used as default; this can be changed by the setInterpolation() method.

For strike dependence, see **BlackVarianceSurface**(p. 218).

Public Member Functions

- **BlackVarianceCurve** (const **Date** &referenceDate, const std::vector< **Date** > &dates, const std::vector< double > &blackVolCurve, const **DayCounter** &dayCounter=**DayCounters::Actual365**())

BlackVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Modifiers

- template<class Traits> void **setInterpolation** ()

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **blackVarianceImpl** (**Time** t, double, bool extrapolate=false) const
  implements the actual Black variance calculation in derived classes

11.55.2 Member Function Documentation

11.55.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

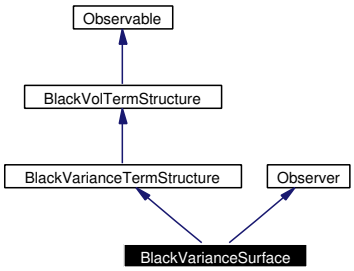
The documentation for this class was generated from the following files:

- ql/Volatilities/**blackvariancecurve.hpp**
- ql/Volatilities/**blackvariancecurve.cpp**

11.56 BlackVarianceSurface Class Reference

`#include <blackvariancesurface.hpp>`

Inheritance diagram for BlackVarianceSurface:



11.56.1 Detailed Description

Black volatility surface modelled as variance surface.

This class calculates time/strike dependent Black volatilities using as input a matrix of Black volatilities observed in the market.

The calculation is performed interpolating on the variance surface. Bilinear interpolation is used as default; this can be changed by the setInterpolation() method.

Public Types

- enum **Extrapolation** { **ConstantExtrapolation**, **InterpolatorDefaultExtrapolation** }

Public Member Functions

- **BlackVarianceSurface** (const **Date** &referenceDate, const std::vector< **Date** > &dates, const std::vector< double > &strikes, const **Math::Matrix** &blackVolMatrix, Extrapolation lowerExtrapolation=InterpolatorDefaultExtrapolation, Extrapolation upperExtrapolation=InterpolatorDefaultExtrapolation, const **DayCounter** &dayCounter=**DayCounters::Actual365**())

BlackVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Modifiers

- template<class Traits> void **setInterpolation** ()

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **blackVarianceImpl** (**Time** t, double strike, bool extrapolate=false) const
  implements the actual Black variance calculation in derived classes

11.56.2 Member Function Documentation

11.56.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

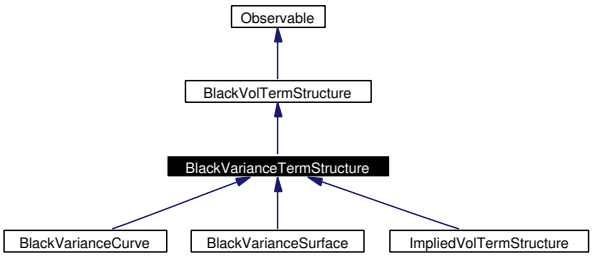
The documentation for this class was generated from the following files:

- ql/Volatilities/**blackvariancesurface.hpp**
- ql/Volatilities/**blackvariancesurface.cpp**

11.57 BlackVarianceTermStructure Class Reference

`#include <voltermstructure.hpp>`

Inheritance diagram for BlackVarianceTermStructure:



11.57.1 Detailed Description

Black Variance term structure.

This abstract class acts as an adapter to VolTermStructure allowing the programmer to implement only the `blackVarianceImpl(Time, double, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- double **blackVolImpl** (**Time** maturity, double strike, bool extrapolate=false) const

11.57.2 Member Function Documentation

11.57.2.1 double blackVolImpl (Time *maturity*, double *strike*, bool *extrapolate* = false) const [protected, virtual]

Returns the volatility for the given strike and date calculating it from the variance.

Implements **BlackVolTermStructure** (p. 223).

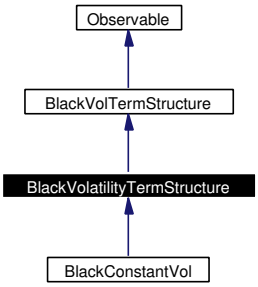
The documentation for this class was generated from the following files:

- ql/**voltermstructure.hpp**
- ql/**voltermstructure.cpp**

11.58 BlackVolatilityTermStructure Class Reference

`#include <voltermstructure.hpp>`

Inheritance diagram for BlackVolatilityTermStructure:



11.58.1 Detailed Description

Black Volatility term structure.

This abstract class acts as an adapter to **BlackVolTermStructure**(p. 222) allowing the programmer to implement only the `blackVolImpl(Time, double, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- double **blackVarianceImpl** (**Time** maturity, double strike, bool extrapolate=false) const

11.58.2 Member Function Documentation

11.58.2.1 double blackVarianceImpl (Time *maturity*, double *strike*, bool *extrapolate* = false) const [protected, virtual]

Returns the variance for the given strike and date calculating it from the volatility.

Implements **BlackVolTermStructure** (p. 223).

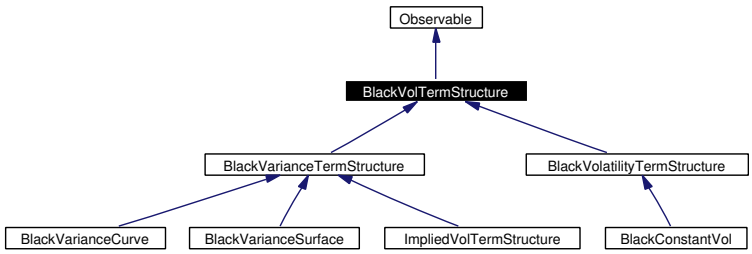
The documentation for this class was generated from the following files:

- ql/**voltermstructure.hpp**
- ql/**voltermstructure.cpp**

11.59 BlackVolTermStructure Class Reference

`#include <voltermstructure.hpp>`

Inheritance diagram for BlackVolTermStructure:



11.59.1 Detailed Description

Black Volatility Term structure.

This abstract class defines the interface of concrete Black volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis

Public Member Functions

Black Volatility

- double **blackVol** (const **Date** &maturity, double strike, bool extrapolate=false) const
  present (a.k.a spot) volatility
- double **blackVol** (**Time** maturity, double strike, bool extrapolate=false) const
  present (a.k.a spot) volatility
- double **blackVariance** (const **Date** &maturity, double strike, bool extrapolate=false) const
  present (a.k.a spot) variance
- double **blackVariance** (**Time** maturity, double strike, bool extrapolate=false) const
  present (a.k.a spot) variance
- double **blackForwardVol** (const **Date** &date1, const **Date** &date2, double strike, bool extrapolate=false) const
  future (a.k.a. forward) volatility
- double **blackForwardVol** (**Time** time1, **Time** time2, double strike, bool extrapolate=false) const
  future (a.k.a. forward) volatility
- double **blackForwardVariance** (const **Date** &date1, const **Date** &date2, double strike, bool extrapolate=false) const

future (a.k.a. forward) variance

- double **blackForwardVariance** (**Time** time1, **Time** time2, double strike, bool extrapolate=false) const
  future (a.k.a. forward) variance

Dates

- virtual **Date referenceDate** () const =0
  returns the reference date for which t=0
- virtual **DayCounter dayCounter** () const =0
  returns the day counter
- virtual **Date maxDate** () const =0
  the latest date for which the term structure can return vols
- **Time maxTime** () const
  the latest time for which the term structure can return vols

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **blackVarianceImpl** (**Time** t, double strike, bool extrapolate=false) const =0
  implements the actual Black variance calculation in derived classes
- virtual double **blackVolImpl** (**Time** t, double strike, bool extrapolate=false) const =0
  implements the actual Black vol calculation in derived classes

The documentation for this class was generated from the following files:

- ql/**voltermstructure.hpp**
- ql/**voltermstructure.cpp**

11.60 BoundaryCondition Class Template Reference

`#include <boundarycondition.hpp>`

11.60.1 Detailed Description

template<class Operator> class QuantLib::FiniteDifferences::BoundaryCondition< Operator >

Abstract boundary condition class for finite difference problems.

Public Types

- typedef Operator **operatorType**
- typedef Operator::arrayType **arrayType**
- enum **Side** { **None**, **Upper**, **Lower** }

Public Member Functions

- virtual void **applyBeforeApplying** (operatorType &) const =0
- virtual void **applyAfterApplying** (arrayType &) const =0
- virtual void **applyBeforeSolving** (operatorType &, arrayType &rhs) const =0
- virtual void **applyAfterSolving** (arrayType &) const =0
- virtual void **setTime** (**Time** t)=0

11.60.2 Member Enumeration Documentation

11.60.2.1 enum Side

Todo

Generalize for n-dimensional conditions

11.60.3 Member Function Documentation

11.60.3.1 virtual void applyBeforeApplying (operatorType &) const [pure virtual]

This method modifies an operator L before it is applied to an array u so that $v = Lu$ will satisfy the given condition.

11.60.3.2 virtual void applyAfterApplying (arrayType &) const [pure virtual]

This method modifies an array u so that it satisfies the given condition.

11.60.3.3 virtual void applyBeforeSolving (operatorType &, arrayType & *rhs*) const [pure virtual]

This method modifies an operator L before the linear system $Lu' = u$ is solved so that u' will satisfy the given condition.

11.60.3.4 virtual void applyAfterSolving (arrayType &) const [pure virtual]

This method modifies an array u so that it satisfies the given condition.

11.60.3.5 virtual void setTime (Time *t*) [pure virtual]

This method sets the current time for time-dependent boundary conditions.

Implemented in **NeumannBC** (p. 511), and **DirichletBC** (p. 300).

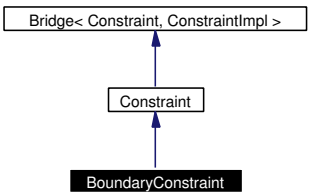
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**boundarycondition.hpp**

11.61 BoundaryConstraint Class Reference

`#include <constraint.hpp>`

Inheritance diagram for BoundaryConstraint:



11.61.1 Detailed Description

Constraint(p. 270) imposing all arguments to be in [low,high].

Public Member Functions

- **BoundaryConstraint** (double low, double high)

The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.62 BoxMullerGaussianRng Class Template Reference

`#include <boxmullergaussianrng.hpp>`

11.62.1 Detailed Description

template<class RNG> class QuantLib::RandomNumbers::BoxMullerGaussianRng< RNG >

Gaussian random number generator.

It uses the well-known Box-Muller transformation to return a normal distributed Gaussian deviate with average 0.0 and standard deviation of 1.0, from a uniform deviate in (0,1) supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **BoxMullerGaussianRng** (const RNG &uniformGenerator)
- **BoxMullerGaussianRng** (long seed=0)
- **sample_type next** () const

  returns next sample from the Gaussian distribution

11.62.2 Constructor & Destructor Documentation

11.62.2.1 BoxMullerGaussianRng (long *seed* = 0) [explicit]

Deprecated
initialize with a random number generator instead.

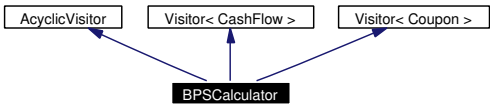
The documentation for this class was generated from the following file:

- ql/RandomNumbers/**boxmullergaussianrng.hpp**

11.63 BPSCalculator Class Reference

`#include <basispointsensitivity.hpp>`

Inheritance diagram for BPSCalculator:



11.63.1 Detailed Description

basis point sensitivity (BPS) calculator

Instances of this class accumulate the BPS of each cash flow they visit, returning the sum through their result() method.

Public Member Functions

- **BPSCalculator** (const **RelinkableHandle**< **TermStructure** > &ts)
- double **result** () const

Visitor interface

- virtual void **visit** (**Coupon** &)
- virtual void **visit** (**CashFlow** &)

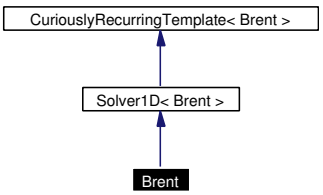
The documentation for this class was generated from the following files:

- ql/CashFlows/**basispointsensitivity.hpp**
- ql/CashFlows/**basispointsensitivity.cpp**

11.64 Brent Class Reference

`#include <brent.hpp>`

Inheritance diagram for Brent:



11.64.1 Detailed Description

Brent 1-D solver

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

The documentation for this class was generated from the following file:

- ql/Solvers1D/**brent.hpp**

11.65 Bridge Class Template Reference

`#include <bridge.hpp>`

11.65.1 Detailed Description

template<class T, class T_impl> class QuantLib::Patterns::Bridge< T, T_impl >

The **Bridge**(p. 230) pattern made explicit.

The typical use of this class is:

```
class FooImpl;
class Foo : public Bridge<Foo,FooImpl> {
    ...
};
```

which makes it possible to pass instances of class Foo by value while retaining polymorphic behavior.

Public Types

- typedef T_impl **Impl**

Public Member Functions

- bool **isNull** () const

Protected Member Functions

- **Bridge** (const **Handle**< Impl > &impl=**Handle**< Impl >())

Protected Attributes

- **Handle**< Impl > **impl_**

The documentation for this class was generated from the following file:

- ql/Patterns/**bridge.hpp**

11.66 BrownianBridge Class Template Reference

`#include <brownianbridge.hpp>`

11.66.1 Detailed Description

template<class GSG> class QuantLib::MonteCarlo::BrownianBridge< GSG >

Builds Wiener process paths using Gaussian variates.

Builds Wiener process paths using Gaussian variates

For more details: "Monte Carlo Methods in Finance" by P. Jäckel, section 10.8.3

Public Types

- typedef **Sample**< std::vector< double > > **sample_type**

Public Member Functions

- **BrownianBridge** (GSG generator)
  normalised (unit time, unit variance) Wiener process paths
- **BrownianBridge** (**Time** length, **Size** timeSteps, GSG generator)
  unit variance Wiener process paths
- **BrownianBridge** (const **TimeGrid** &timeGrid, GSG generator)
  unit variance Wiener process paths
- **BrownianBridge** (const std::vector< double > &sigma, const **TimeGrid** &timeGrid, GSG generator)
  general Wiener process paths
- **BrownianBridge** (const **Handle**< **BlackVolTermStructure** > &blackVol, const **TimeGrid** &timeGrid, GSG generator)

inspectors

- const **sample_type** & **next** () const
- const **sample_type** & **antithetic** () const
- **Size size** () const
- const **TimeGrid** & **timeGrid** () const

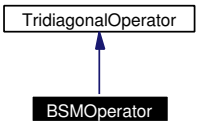
The documentation for this class was generated from the following file:

- ql/MonteCarlo/**brownianbridge.hpp**

11.67 BSMOperator Class Reference

`#include <bsmoperator.hpp>`

Inheritance diagram for BSMOperator:



11.67.1 Detailed Description

Black-Scholes-Merton differential operator.

Public Member Functions

- **BSMOperator** (**Size** size, double dx, double r, double q, double sigma)

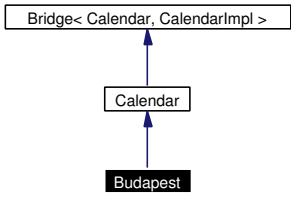
The documentation for this class was generated from the following files:

- ql/FiniteDifferences/**bsmoperator.hpp**
- ql/FiniteDifferences/**bsmoperator.cpp**

11.68 Budapest Class Reference

`#include <budapest.hpp>`

Inheritance diagram for Budapest:



11.68.1 Detailed Description

Budapest calendar

Holidays:

- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

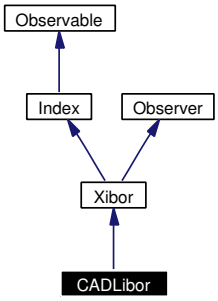
The documentation for this class was generated from the following file:

- ql/Calendars/**budapest.hpp**

11.69 CADLibor Class Reference

`#include <cadlibor.hpp>`

Inheritance diagram for CADLibor:



11.69.1 Detailed Description

CAD Libor index (Also known as CDOR)

Todo
check settlement days

Public Member Functions

- **CADLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual365**())

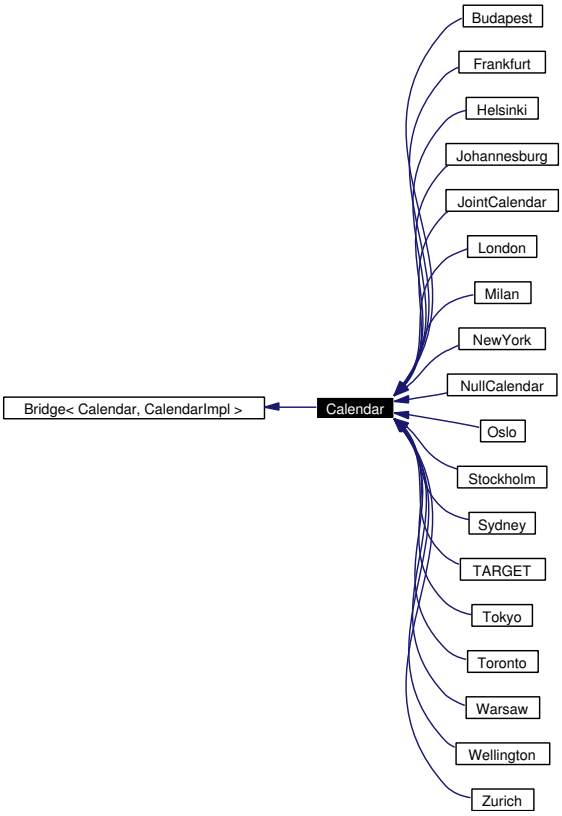
The documentation for this class was generated from the following file:

- ql/Indexes/**cadlibor.hpp**

11.70 Calendar Class Reference

`#include <calendar.hpp>`

Inheritance diagram for Calendar:



11.70.1 Detailed Description

calendar class

This class provides methods for determining whether a date is a business day or a holiday for a given market, and for incrementing/decrementing a date of a given number of business days.

The Bridge pattern is used to provide the base behavior of the calendar, namely, to determine whether a date is a business day.

Public Member Functions

- **Calendar** ()

Calendar interface

- std::string **name** () const
  Returns the name of the calendar.
- bool **isBusinessDay** (const **Date** &d) const

- bool **isEndOfMonth** (const **Date** &d) const
- bool **isHoliday** (const **Date** &d) const
- **Date roll** (const **Date** &, **RollingConvention** convention=Following, const **Date** &origin=**Date**()) const
- **Date advance** (const **Date** &, int n, **TimeUnit** unit, **RollingConvention** convention=Following) const
- **Date advance** (const **Date** &date, const **Period** &period, **RollingConvention** convention) const

Protected Member Functions

- **Calendar** (const **Handle**< **CalendarImpl** > &impl)

Related Functions

(Note that these are not member functions.)

- bool **operator==** (const **Calendar** &, const **Calendar** &)
- bool **operator!=** (const **Calendar** &, const **Calendar** &)

11.70.2 Constructor & Destructor Documentation

11.70.2.1 Calendar ()

This default constructor returns a calendar with a null implementation, which is therefore unusable except as a placeholder.

11.70.2.2 Calendar (const Handle< CalendarImpl > & *impl*) [protected]

This protected constructor will only be invoked by derived classes which define a given **Calendar**(p. 235) implementation

11.70.3 Member Function Documentation

11.70.3.1 std::string name () const

Returns the name of the calendar.

Warning:
This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

11.70.3.2 bool isBusinessDay (const Date & *d*) const

Returns `true` iff the date is a business day for the given market.

11.70.3.3 bool isEndOfMonth (const Date & *d*) const

Returns `true` iff the date is last business day for the month in given market.

11.70.3.4 bool isHoliday (const Date & *d*) const

Returns `true` iff the date is a holiday for the given market.

11.70.3.5 Date roll (const Date &, RollingConvention *convention* = Following, const Date & *origin* = Date()) const

Returns the next business day on the given market with respect to the given date and convention.

11.70.3.6 Date advance (const Date &, int *n*, TimeUnit *unit*, RollingConvention *convention* = Following) const

Advances the given date of the given number of business days and returns the result.

Note:
The input date is not modified.

11.70.3.7 Date advance (const Date & *date*, const Period & *period*, RollingConvention *convention*) const

Advances the given date as specified by the given period and returns the result.

Note:
The input date is not modified.

11.70.4 Friends And Related Function Documentation

11.70.4.1 bool operator== (const Calendar &, const Calendar &) [related]

Returns `true` iff the two calendars belong to the same derived class.

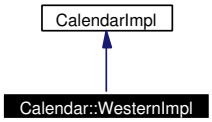
The documentation for this class was generated from the following files:

- ql/**calendar.hpp**
- ql/**calendar.cpp**

11.71 Calendar::WesternImpl Class Reference

`#include <calendar.hpp>`

Inheritance diagram for Calendar::WesternImpl:



11.71.1 Detailed Description

partial implementation providing the means of determining the Easter Monday for a given year.

Static Protected Member Functions

- **Day easterMonday** (**Year** y)
  expressed relative to first day of year

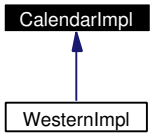
The documentation for this class was generated from the following files:

- ql/**calendar.hpp**
- ql/**calendar.cpp**

11.72 CalendarImpl Class Reference

`#include <calendar.hpp>`

Inheritance diagram for CalendarImpl:



11.72.1 Detailed Description

abstract base class for calendar implementations

Public Member Functions

- virtual std::string **name** () const =0
- virtual bool **isBusinessDay** (const **Date** &) const =0

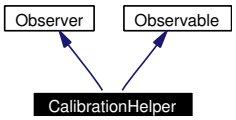
The documentation for this class was generated from the following file:

- ql/**calendar.hpp**

11.73 CalibrationHelper Class Reference

`#include <calibrationhelper.hpp>`

Inheritance diagram for CalibrationHelper:



11.73.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

- **CalibrationHelper** (const **RelinkableHandle**< **MarketElement** > &volatility, const **RelinkableHandle**< **TermStructure** > &termStructure)
- void **update** ()
- double **marketValue** ()

  returns the actual price of the instrument (from volatility)

- virtual double **modelValue** ()=0

  returns the price of the instrument according to the model

- virtual double **calibrationError** ()

  returns the error resulting from the model valuation

- virtual void **addTimesTo** (std::list< **Time** > &times) const =0
- double **impliedVolatility** (double targetValue, double accuracy, **Size** maxEvaluations, double minVol, double maxVol) const

  Black volatility implied by the model.

- virtual double **blackPrice** (double volatility) const =0

  Black price given a volatility.

- void **setPricingEngine** (const **Handle**< **PricingEngine** > &engine)

Protected Attributes

- double **marketValue_**
- **RelinkableHandle**< **MarketElement** > **volatility_**
- **RelinkableHandle**< **TermStructure** > **termStructure_**
- **Handle**< **BlackModel** > **blackModel_**
- **Handle**< **PricingEngine** > **engine_**

11.73.2 Member Function Documentation

11.73.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

The documentation for this class was generated from the following files:

- ql/ShortRateModels/**calibrationhelper.hpp**
- ql/ShortRateModels/**calibrationhelper.cpp**

11.74 CalibrationSet Class Reference

`#include <calibrationhelper.hpp>`

11.74.1 Detailed Description

Set of calibration instruments.

For the moment, this is just here to facilitate the assignment of a pricing engine to a set of calibration helpers

Public Member Functions

- void **setPricingEngine** (const **Handle**< **PricingEngine** > &engine)

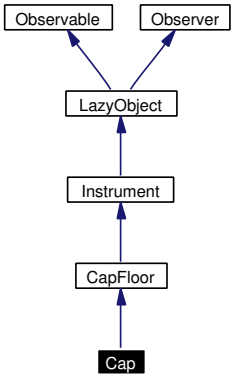
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**calibrationhelper.hpp**

11.75 Cap Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for Cap:



11.75.1 Detailed Description

Concrete cap class.

Public Member Functions

- **Cap** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< **Rate** > &exerciseRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

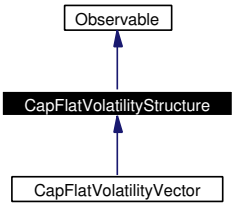
The documentation for this class was generated from the following file:

- ql/Instruments/**capfloor.hpp**

11.76 CapFlatVolatilityStructure Class Reference

`#include <capvolstructures.hpp>`

Inheritance diagram for CapFlatVolatilityStructure:



11.76.1 Detailed Description

Cap/floor flat volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

- virtual **Date todaysDate** () const =0
  returns today's date
- virtual **Date settlementDate** () const =0
  returns the settlement date
- virtual **DayCounter dayCounter** () const =0
  returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &end, **Rate** strike) const
  returns the volatility for a given end date and strike rate
- double **volatility** (const **Period** &length, **Rate** strike) const
  returns the volatility for a given cap/floor length and strike rate
- double **volatility** (**Time** t, **Rate** strike) const
  returns the volatility for a given end time and strike rate

Protected Member Functions

- virtual double **volatilityImpl** (**Time** length, **Rate** strike) const =0
  implements the actual volatility calculation in derived classes

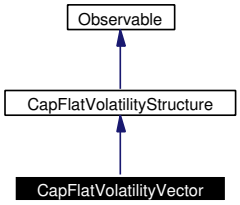
The documentation for this class was generated from the following file:

- ql/**capvolstructures.hpp**

11.77 CapFlatVolatilityVector Class Reference

`#include <capflatvolvector.hpp>`

Inheritance diagram for CapFlatVolatilityVector:



11.77.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

This class provides the at-the-money volatility for a given cap by interpolating a volatility vector whose elements are the market volatilities of a set of caps/floors with given length.

Todo

Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Public Member Functions

- **CapFlatVolatilityVector** (const **Date** &todaysDate, const **Calendar** &calendar, int settlementDays, const std::vector< **Period** > &lengths, const std::vector< double > &volatilities, const **DayCounter** &dayCounter=**DayCounters::Thirty360**())
- **Date todaysDate** () const

  returns today's date

- **Date settlementDate** () const

  returns the settlement date

- **DayCounter dayCounter** () const

  returns the day counter used for internal date/time conversions

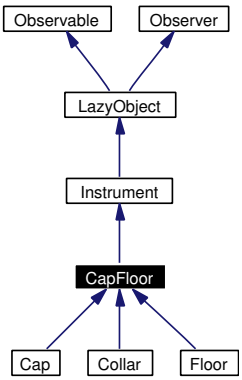
The documentation for this class was generated from the following file:

- ql/Volatilities/**capflatvolvector.hpp**

11.78 CapFloor Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for CapFloor:



11.78.1 Detailed Description

Base class for cap-like instruments.

Public Types

- enum **Type** { **Cap**, **Floor**, **Collar** }

Public Member Functions

- **CapFloor** (Type type, const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< **Rate** > &capRates, const std::vector< **Rate** > &floorRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)
- void **setupArguments** (**Arguments** *) const
- double **impliedVolatility** (double price, double accuracy=1.0e-4, **Size** maxEvaluations=100, double minVol=QL_MIN_VOLATILITY, double maxVol=QL_MAX_VOLATILITY) const

  implied term volatility

Instrument interface

- bool **isExpired** () const

  returns whether the instrument is still tradable.

Inspectors

- Type **type** () const
- const std::vector< **Handle**< **CashFlow** > > & **leg** () const
- const std::vector< **Rate** > & **capRates** () const
- const std::vector< **Rate** > & **floorRates** () const

11.78.2 Member Function Documentation

11.78.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

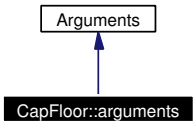
The documentation for this class was generated from the following files:

- ql/Instruments/**capfloor.hpp**
- ql/Instruments/**capfloor.cpp**

11.79 CapFloor::arguments Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for CapFloor::arguments:



11.79.1 Detailed Description

arguments for cap/floor calculation

Public Member Functions

- void **validate** () const

Public Attributes

- CapFloor::Type **type**
- std::vector< **Time** > **startTimes**
- std::vector< **Time** > **fixingTimes**
- std::vector< **Time** > **endTimes**
- std::vector< **Time** > **accrualTimes**
- std::vector< **Rate** > **capRates**
- std::vector< **Rate** > **floorRates**
- std::vector< **Rate** > **forwards**
- std::vector< double > **nominals**

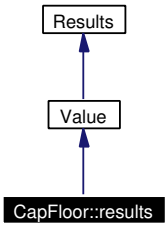
The documentation for this class was generated from the following files:

- ql/Instruments/**capfloor.hpp**
- ql/Instruments/**capfloor.cpp**

11.80 CapFloor::results Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for CapFloor::results:



11.80.1 Detailed Description

results from cap/floor calculation

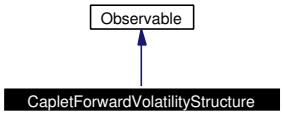
The documentation for this class was generated from the following file:

- ql/Instruments/**capfloor.hpp**

11.81 CapletForwardVolatilityStructure Class Reference

`#include <capvolstructures.hpp>`

Inheritance diagram for CapletForwardVolatilityStructure:



11.81.1 Detailed Description

Caplet/floorlet forward volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

- virtual **Date todaysDate** () const =0
  returns today's date
- virtual **Date settlementDate** () const =0
  returns the settlement date
- virtual **DayCounter dayCounter** () const =0
  returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &start, **Rate** strike) const
  returns the volatility for a given start date and strike rate
- double **volatility** (**Time** t, **Rate** strike) const
  returns the volatility for a given start time and strike rate

Protected Member Functions

- virtual double **volatilityImpl** (**Time** length, **Rate** strike) const =0
  implements the actual volatility calculation in derived classes

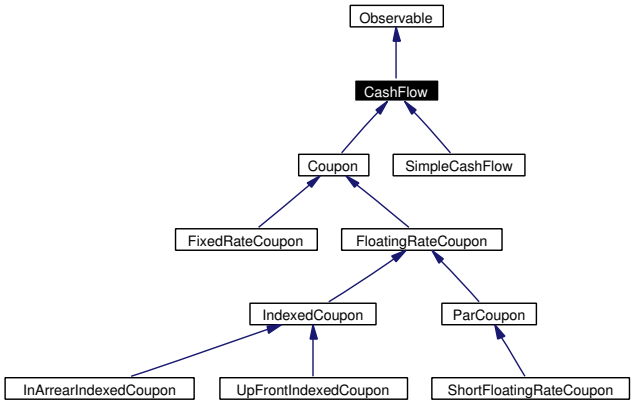
The documentation for this class was generated from the following file:

- ql/**capvolstructures.hpp**

11.82 CashFlow Class Reference

`#include <cashflow.hpp>`

Inheritance diagram for CashFlow:



11.82.1 Detailed Description

Base class for cash flows.

This class is purely virtual and acts as a base class for the actual cash flow implementations.

Public Member Functions

CashFlow interface

- virtual double **amount** () const =0
  returns the amount of the cash flow
- virtual **Date date** () const =0
  returns the date at which the cash flow is settled

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.82.2 Member Function Documentation

11.82.2.1 virtual double amount () const [pure virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implemented in **FixedRateCoupon** (p. 354), **IndexedCoupon** (p. 420), **ParCoupon** (p. 542), **ShortFloatingRateCoupon** (p. 594), and **SimpleCashFlow** (p. 596).

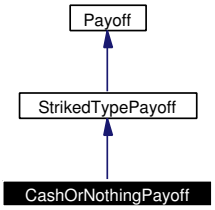
The documentation for this class was generated from the following file:

- ql/**cashflow.hpp**

11.83 CashOrNothingPayoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for CashOrNothingPayoff:



11.83.1 Detailed Description

Binary(p. 195) Cash-Or-Nothing option payoff.

Public Member Functions

- **CashOrNothingPayoff** (Option::Type type, double strike, double cashPayoff)
- double **operator()** (double price) const

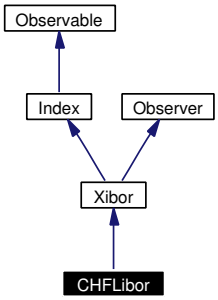
The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.84 CHFLibor Class Reference

`#include <chflibor.hpp>`

Inheritance diagram for CHFLibor:



11.84.1 Detailed Description

CHF Libor index (Also known as ZIBOR)

Todo
check settlement days and day-count

Public Member Functions

- **CHFLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual360**())

The documentation for this class was generated from the following file:

- ql/Indexes/**chflibor.hpp**

11.85 CLGaussianRng Class Template Reference

`#include <centrallimitgaussianrng.hpp>`

11.85.1 Detailed Description

template<class RNG> class QuantLib::RandomNumbers::CLGaussianRng< RNG >

Gaussian random number generator.

It uses the well-known fact that the sum of 12 uniform deviate in (-.5,.5) is approximately a Gaussian deviate with average 0 and standard deviation 1. The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **CLGaussianRng** (const RNG &uniformGenerator)
- **CLGaussianRng** (long seed=0)
- **sample_type next** () const

  returns next sample from the Gaussian distribution

11.85.2 Constructor & Destructor Documentation

11.85.2.1 CLGaussianRng (long *seed* = 0) [explicit]

Deprecated
initialize with a random number generator instead.

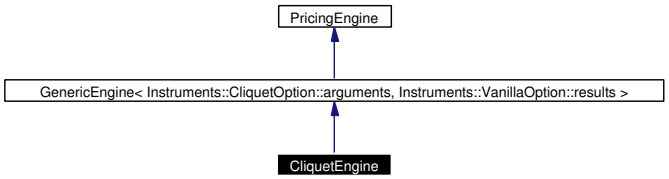
The documentation for this class was generated from the following file:

- ql/RandomNumbers/**centrallimitgaussianrng.hpp**

11.86 CliquetEngine Class Reference

`#include <cliquetengines.hpp>`

Inheritance diagram for CliquetEngine:



11.86.1 Detailed Description

Cliquet engine base class.

The documentation for this class was generated from the following file:

- ql/PricingEngines/**cliquetengines.hpp**

11.87 CliquetOption Class Reference

`#include <cliquetoption.hpp>`

11.87.1 Detailed Description

currently just a container for arguments and results

The documentation for this class was generated from the following file:

- ql/Instruments/cliquetoption.hpp

11.88 CliquetOption Class Reference

`#include <cliquetoption.hpp>`

11.88.1 Detailed Description

cliquet (Ratchet) option

A cliquet option, also known as Ratchet option, is a series of forward-starting (a.k.a. deferred strike) options where the strike for each forward start option is set equal to a fixed percentage of the spot price at the beginning of each period.

In the particular case in which only two dates are given the cliquet option is the same as a forward-starting option starting at the first date and expiring at the second date.

Public Member Functions

- **CliquetOption** (Option::Type type, double underlying, double moneyness, const std::vector< **Spread** > &dividendYield, const std::vector< **Rate** > &riskFreeRate, const std::vector< **Time** > &times, const std::vector< double > &volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const

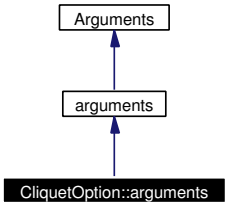
The documentation for this class was generated from the following files:

- ql/Pricers/cliquetoption.hpp
- ql/Pricers/**cliquetoption.cpp**

11.89 CliquetOption::arguments Class Reference

`#include <cliquetoption.hpp>`

Inheritance diagram for CliquetOption::arguments:



11.89.1 Detailed Description

arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- double **moneyness**
- double **accruedCoupon**
- double **lastFixing**
- double **localCap**
- double **localFloor**
- double **globalCap**
- double **globalFloor**
- std::vector< **Date** > **resetDates**

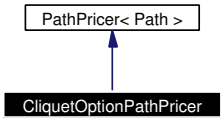
The documentation for this class was generated from the following file:

- ql/Instruments/cliquetoption.hpp

11.90 CliquetOptionPathPricer Class Reference

`#include <cliquetoptionpathpricer.hpp>`

Inheritance diagram for CliquetOptionPathPricer:



11.90.1 Detailed Description

path pricer for cliquet options

Public Member Functions

- **CliquetOptionPathPricer** (Option::Type type, double underlying, double moneyness, double accruedCoupon, double lastFixing, double localCap, double localFloor, double globalCap, double globalFloor, const **RelinkableHandle**< **TermStructure** > &riskFreeTS)
- double **operator()** (const **Path** &path) const

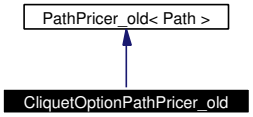
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**cliquetoptionpathpricer.hpp**
- ql/MonteCarlo/**cliquetoptionpathpricer.cpp**

11.91 CliquetOptionPathPricer_old Class Reference

`#include <cliquetoptionpathpricer.hpp>`

Inheritance diagram for CliquetOptionPathPricer_old:



11.91.1 Detailed Description

path pricer for cliquet options

path pricer for cliquet options

Deprecated
use **CliquetOptionPathPricer**(p. 261) instead

Public Member Functions

- **CliquetOptionPathPricer_old** (Option::Type type, double underlying, double moneyness, double accruedCoupon, double lastFixing, double localCap, double localFloor, double globalCap, double globalFloor, const std::vector< **DiscountFactor** > &discounts, bool redemptionOnly, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

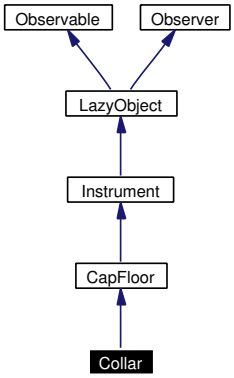
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**cliquetoptionpathpricer.hpp**
- ql/MonteCarlo/**cliquetoptionpathpricer.cpp**

11.92 Collar Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for Collar:



11.92.1 Detailed Description

Concrete cap class.

Public Member Functions

- **Collar** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< **Rate** > &capRates, const std::vector< **Rate** > &floorRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

The documentation for this class was generated from the following file:

- ql/Instruments/**capfloor.hpp**

11.93 combining_iterator Class Template Reference

`#include <combiningiterator.hpp>`

11.93.1 Detailed Description

template<class Iterator, class Function> class QuantLib::Utilities::combining_iterator< Iterator, Function >

Iterator mapping a function to a set of underlying sequences.

This iterator advances a set of underlying iterators and returns the values obtained by applying a function to the sets of values such iterators point to.

This class was implemented based on Christopher Baus and Thomas Becker, *Custom Iterators for the STL*, included in the proceedings of the First Workshop on C++ Template Programming, Erfurt, Germany, 2000 (`http://www.oonumerics.org/tmpw00/`)

Public Types

- typedef Function::result_type **value_type**
- typedef const Function::result_type * **pointer**
- typedef const Function::result_type & **reference**

Public Member Functions

- template<class IteratorCollectionIterator> **combining_iterator** (IteratorCollectionIterator it1, IteratorCollectionIterator it2, Function f)

Dereferencing

- reference **operator** * () const
- pointer **operator** → () const

Random access

- value_type **operator[ ]** (difference_type n) const

Increment and decrement

- **combining_iterator** & **operator++** ()
- **combining_iterator operator++** (int)
- **combining_iterator** & **operator–** ()
- **combining_iterator operator–** (int)
- **combining_iterator** & **operator+=** (difference_type n)
- **combining_iterator** & **operator-=** (difference_type n)
- **combining_iterator operator+** (difference_type n) const
- **combining_iterator operator-** (difference_type n) const

Difference

- difference_type **operator-** (const **combining_iterator**< Iterator, Function > &rhs) const

Comparisons

- bool **operator==** (const **combining_iterator**< Iterator, Function > &rhs) const
- bool **operator!=** (const **combining_iterator**< Iterator, Function > &rhs) const

Public Attributes

- typedef< Iterator >::difference_type **difference_type**

Related Functions

(Note that these are not member functions.)

- **combining_iterator**< typename 1< It >::value_type, Function > **make_combining_iterator** (It it1, It it2, Function f)

  helper function to create combining iterators

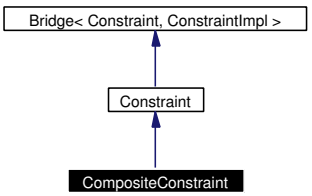
The documentation for this class was generated from the following file:

- ql/Utilities/**combiningiterator.hpp**

11.94 CompositeConstraint Class Reference

`#include <constraint.hpp>`

Inheritance diagram for CompositeConstraint:



11.94.1 Detailed Description

Constraint(p. 270) enforcing both given sub-constraints.

Public Member Functions

- **CompositeConstraint** (const **Constraint** &c1, const **Constraint** &c2)

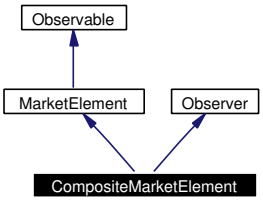
The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.95 CompositeMarketElement Class Template Reference

`#include <marketelement.hpp>`

Inheritance diagram for CompositeMarketElement:



11.95.1 Detailed Description

template<class BinaryFunction> class QuantLib::CompositeMarketElement< BinaryFunction >

market element whose value depends on two other market element

Public Member Functions

- **CompositeMarketElement** (const **RelinkableHandle**< **MarketElement** > &element1, const **RelinkableHandle**< **MarketElement** > &element2, const BinaryFunction &f)

Market element interface

- double **value** () const
  returns the current value

Observer interface

- void **update** ()

11.95.2 Member Function Documentation

11.95.2.1 void update () `[virtual]`

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

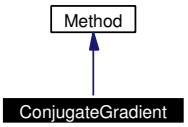
The documentation for this class was generated from the following file:

- ql/**marketelement.hpp**

11.96 ConjugateGradient Class Reference

`#include <conjugategradient.hpp>`

Inheritance diagram for ConjugateGradient:



11.96.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

User has to provide line-search method and optimization end criteria

search direction $d_i = -f'(x_i) + c_i * d_{i-1}$ where $c_i = ||f'(x_i)||^2/||f'(x_{i-1})||^2$ and $d_1 = -f'(x_1)$

Public Member Functions

- **ConjugateGradient** ()
  default constructor
- **ConjugateGradient** (const **Handle**< **LineSearch** > &lineSearch)
- virtual ~**ConjugateGradient** ()
  destructor
- virtual void **minimize** (const **Problem** &P) const
  minimize the optimization problem P

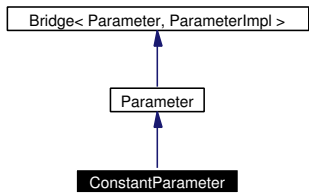
The documentation for this class was generated from the following files:

- ql/Optimization/**conjugategradient.hpp**
- ql/Optimization/**conjugategradient.cpp**

11.97 ConstantParameter Class Reference

`#include <parameter.hpp>`

Inheritance diagram for ConstantParameter:



11.97.1 Detailed Description

Standard constant parameter $a(t) = a$.

Public Member Functions

- **ConstantParameter** (const **Optimization::Constraint** &constraint)
- **ConstantParameter** (double value, const **Optimization::Constraint** &constraint)

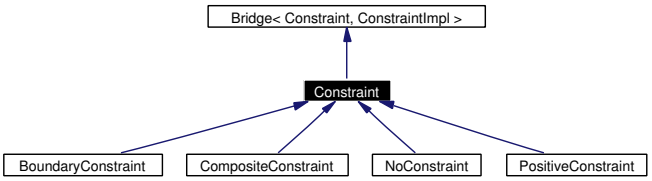
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.98 Constraint Class Reference

`#include <constraint.hpp>`

Inheritance diagram for Constraint:



11.98.1 Detailed Description

Base constraint class.

Public Member Functions

- bool **test** (const **Array** &p) const
- double **update** (**Array** &p, const **Array** &direction, double beta)
- **Constraint** (const **Handle**< **ConstraintImpl** > &impl=**Handle**< **ConstraintImpl** >())

The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.99 ConstraintImpl Class Reference

`#include <constraint.hpp>`

11.99.1 Detailed Description

Base class for **Constraint**(p. 270) implementations.

Public Member Functions

- virtual bool **test** (const **Array** &params) const =0

  Tests if params satisfy the constraint.

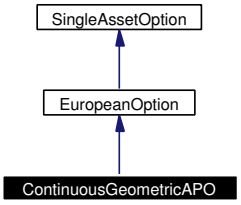
The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.100 ContinuousGeometricAPO Class Reference

`#include <continuousgeometricapo.hpp>`

Inheritance diagram for ContinuousGeometricAPO:



11.100.1 Detailed Description

Continuous Geometric Average Price **Option**(p. 533) (European exercise).

This class implements a continuous geometric average price asian option with european exercise. The formula is from "Option Pricing Formulas", E. G. Haug (1997) pag 96-97

Todo

add Average Strike version and make it backward starting

Public Member Functions

- **ContinuousGeometricAPO** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility)
- double **vega** () const
- double **rho** () const
- **Handle**< **SingleAssetOption** > **clone** () const

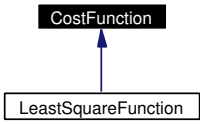
The documentation for this class was generated from the following file:

- ql/Pricers/**continuousgeometricapo.hpp**

11.101 CostFunction Class Reference

`#include <costfunction.hpp>`

Inheritance diagram for CostFunction:



11.101.1 Detailed Description

Cost function abstract class for optimization problem.

Public Member Functions

- virtual double **value** (const **Array** &x) const =0
  method to overload to compute the cost functon value in x
- virtual void **gradient** (**Array** &grad, const **Array** &x) const
  method to overload to compute grad_f, the first derivative of
- virtual double **valueAndGradient** (**Array** &grad, const **Array** &x) const
  method to overload to compute grad_f, the first derivative of
- virtual double **finiteDifferenceEpsilon** () const
  Default epsilon for finite difference method :.

The documentation for this class was generated from the following file:

- ql/Optimization/**costfunction.hpp**

11.102 coupling_iterator Class Template Reference

`#include <couplingiterator.hpp>`

11.102.1 Detailed Description

template<class Iterator1, class Iterator2, class Function> class QuantLib::Utilities::coupling_iterator< Iterator1, Iterator2, Function >

Iterator mapping a function to a pair of underlying sequences.

This iterator advances two underlying iterators and returns the values obtained by applying a function to the two values such iterators point to.

Public Types

- typedef Function::result_type **value_type**
- typedef const Function::result_type * **pointer**
- typedef const Function::result_type & **reference**

Public Member Functions

- **coupling_iterator** (Iterator1 it1, Iterator2 it2, Function f)

Dereferencing

- reference **operator** * () const
- pointer **operator** → () const

Random access

- value_type **operator[ ]** (difference_type n) const

Increment and decrement

- **coupling_iterator** & **operator++** ()
- **coupling_iterator operator++** (int)
- **coupling_iterator** & **operator–** ()
- **coupling_iterator operator–** (int)
- **coupling_iterator** & **operator+=** (difference_type n)
- **coupling_iterator** & **operator-=** (difference_type n)
- **coupling_iterator operator+** (difference_type n) const
- **coupling_iterator operator-** (difference_type n) const

Difference

- difference_type **operator-** (const **coupling_iterator** &rhs) const

Comparisons

- bool **operator==** (const **coupling_iterator** &rhs) const
- bool **operator!=** (const **coupling_iterator** &rhs) const

Public Attributes

- typedef< Iterator1 >::difference_type **difference_type**

Related Functions

(Note that these are not member functions.)

- **coupling_iterator**< It1, It2, Function > **make_coupling_iterator** (It1 it1, It2 it2, Function f)
  helper function to create combining iterators

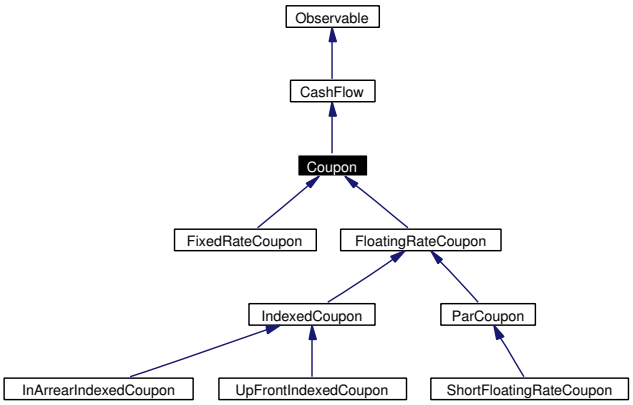
The documentation for this class was generated from the following file:

- ql/Utilities/**couplingiterator.hpp**

11.103 Coupon Class Reference

`#include <coupon.hpp>`

Inheritance diagram for Coupon:



11.103.1 Detailed Description

coupon accruing over a fixed period

This class implements part of the **CashFlow**(p. 252) interface but it is still abstract and provides derived classes with methods for accrual period calculations.

Public Member Functions

- **Coupon** (double nominal, const **Date** &paymentDate, const **Date** &accrualStartDate, const **Date** &accrualEndDate, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

Partial CashFlow interface

- **Date date** () const
  returns the date at which the cash flow is settled

Inspectors

- double **nominal** () const
- const **Date** & **accrualStartDate** () const
  start of the accrual period
- const **Date** & **accrualEndDate** () const
  end of the accrual period
- **Time accrualPeriod** () const
  accrual period as fraction of year

- int **accrualDays** () const
  accrual period in days
- virtual **DayCounter dayCounter** () const =0
  day counter for accrual calculation
- virtual double **accruedAmount** (const **Date** &) const =0
  accrued amount at the given date

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Attributes

- double **nominal_**
- **Date paymentDate_**
- **Date accrualStartDate_**
- **Date accrualEndDate_**
- **Date refPeriodStart_**
- **Date refPeriodEnd_**

11.103.2 Constructor & Destructor Documentation

11.103.2.1 Coupon (double *nominal*, const Date & *paymentDate*, const Date & *accrualStartDate*, const Date & *accrualEndDate*, const Date & *refPeriodStart* = Date(), const Date & *refPeriodEnd* = Date())

Warning:
the coupon does not roll the payment date which must already be a business day.

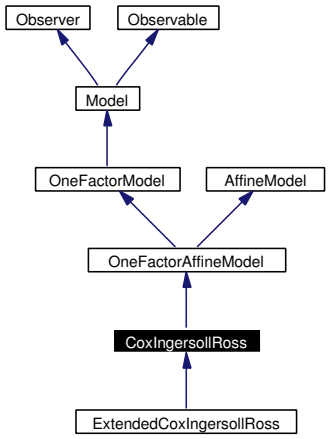
The documentation for this class was generated from the following file:

- ql/CashFlows/**coupon.hpp**

11.104 CoxIngersollRoss Class Reference

`#include <coxingersollross.hpp>`

Inheritance diagram for CoxIngersollRoss:



11.104.1 Detailed Description

Cox-Ingersoll-Ross model class.

This class implements the Cox-Ingersoll-Ross model defined by

$$dr_t = k(\theta - r_t)dt + \sqrt{r_t}\sigma dW_t.$$

Unstable
This class was not tested enough to guarantee its functionality.

Public Member Functions

- **CoxIngersollRoss** (**Rate** r0=0.05, double theta=0.1, double k=0.1, double sigma=0.1)
- virtual double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const
- virtual **Handle**< ShortRateDynamics > **dynamics** () const

  returns the short-rate dynamics

- virtual **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Return by default a trinomial recombining tree.

Protected Member Functions

- double **A** (**Time** t, **Time** T) const
- double **B** (**Time** t, **Time** T) const
- double **theta** () const
- double **k** () const

- double **sigma** () const
- double **x0** () const

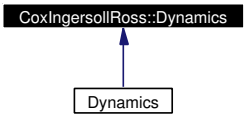
The documentation for this class was generated from the following files:

- ql/ShortRateModels/OneFactorModels/**coxingersollross.hpp**
- ql/ShortRateModels/OneFactorModels/**coxingersollross.cpp**

11.105 CoxIngersollRoss::Dynamics Class Reference

`#include <coxingersollross.hpp>`

Inheritance diagram for CoxIngersollRoss::Dynamics:



11.105.1 Detailed Description

Dynamics(p. 280) of the short-rate under the Cox-Ingersoll-Ross model.

The state variable y_t will here be the square-root of the short-rate. It satisfies the following stochastic equation

$$dy_t = \left[(\frac{k\theta}{2} + \frac{\sigma^2}{8}) \frac{1}{y_t} - \frac{k}{2} y_t \right] d_t + \frac{\sigma}{2} dW_t$$

.

Public Member Functions

- **Dynamics** (double theta, double k, double sigma, double x0)
- virtual double **variable** (**Time** t, **Rate** r) const
- virtual double **shortRate** (**Time** t, double y) const

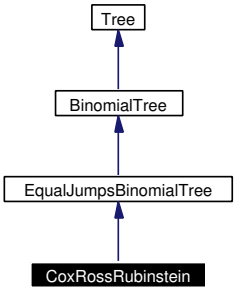
The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**coxingersollross.hpp**

11.106 CoxRossRubinstein Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for CoxRossRubinstein:



11.106.1 Detailed Description

Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

Public Member Functions

- **CoxRossRubinstein** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)

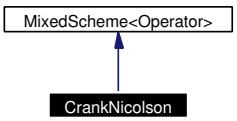
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.107 CrankNicolson Class Template Reference

`#include <cranknicolson.hpp>`

Inheritance diagram for CrankNicolson:



11.107.1 Detailed Description

template<class Operator> class QuantLib::FiniteDifferences::CrankNicolson< Operator >

Crank-Nicolson scheme for finite difference methods.

See sect. **The finite differences framework**(p. 31) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:
The differential operator must be linear for this evolver to work.

Friends

- class **FiniteDifferenceModel< CrankNicolson< Operator > >**

The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**cranknicolson.hpp**

11.108 CubicSpline Class Reference

`#include <interpolationtraits.hpp>`

11.108.1 Detailed Description

cubic-spline interpolation traits

Static Public Member Functions

- template<class I1, class I2> **Handle**< **Interpolation**< I1, I2 > > **make_interpolation** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)
- template<class I1, class I2, class M> **Handle**< **Interpolation2D**< I1, I2, M > > **make_interpolation** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z)

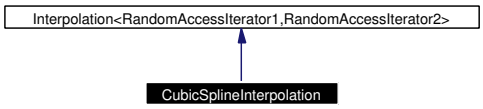
The documentation for this class was generated from the following file:

- ql/Math/**interpolationtraits.hpp**

11.109 CubicSplineInterpolation Class Template Reference

`#include <cubicspline.hpp>`

Inheritance diagram for CubicSplineInterpolation:



11.109.1 Detailed Description

template<class RandomAccessIterator1, class RandomAccessIterator2> class QuantLib::Math::CubicSplineInterpolation< RandomAccessIterator1, RandomAccessIterator2 >

cubic spline interpolation between discrete points

Public Member Functions

- **CubicSplineInterpolation** (const RandomAccessIterator1 &xBegin, const RandomAccessIterator1 &xEnd, const RandomAccessIterator2 &yBegin)
- result_type **operator()** (const argument_type &x, bool allowExtrapolation=false) const
- result_type **derivative** (const argument_type &x, bool allowExtrapolation=false) const
- result_type **secondDerivative** (const argument_type &x, bool allowExtrapolation=false) const

Public Attributes

- typedef< RandomAccessIterator1 >::value_type **argument_type**
- typedef< RandomAccessIterator2 >::value_type **result_type**

11.109.2 Member Function Documentation

11.109.2.1 result_type operator() (const argument_type & *x*, bool *allowExtrapolation* = false) const [virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implements **Interpolation** (p. 428).

The documentation for this class was generated from the following file:

- ql/Math/**cubicspline.hpp**

11.110 CumulativeNormalDistribution Class Reference

`#include <normaldistribution.hpp>`

11.110.1 Detailed Description

Cumulative normal distribution function.

Given x it provides an approximation to the integral of the gaussian normal distribution: formula here ...

For this implementation see M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Dover Publications, New York (1972)

Public Member Functions

- **CumulativeNormalDistribution** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const
- double **derivative** (double x) const

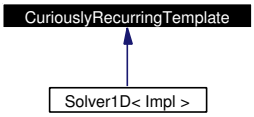
The documentation for this class was generated from the following files:

- ql/Math/**normaldistribution.hpp**
- ql/Math/**normaldistribution.cpp**

11.111 CuriouslyRecurringTemplate Class Template Reference

`#include <curiouslyrecurring.hpp>`

Inheritance diagram for CuriouslyRecurringTemplate:



11.111.1 Detailed Description

template<class Impl> class QuantLib::Patterns::CuriouslyRecurringTemplate< Impl >

Support for the curiously recurring template pattern.

See James O. Coplien. A Curiously Recurring Template Pattern. In Stanley B. Lippman, editor, C++ Gems, 135-144. Cambridge University Press, New York, New York, 1996.

Protected Member Functions

- Impl & **impl** ()
- const Impl & **impl** () const

The documentation for this class was generated from the following file:

- ql/Patterns/**curiouslyrecurring.hpp**

11.112 CurrencyFormatter Class Reference

`#include <dataformatters.hpp>`

11.112.1 Detailed Description

Formats currencies for output.

Static Public Member Functions

- std::string **toString** (**Currency** c)

The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.113 Date Class Reference

`#include <date.hpp>`

11.113.1 Detailed Description

Concrete date class.

This class provides methods to inspect dates as well as methods and operators which implement a limited date algebra (increasing and decreasing dates, and calculating their difference).

Public Member Functions

constructors

- **Date** ()
  Default constructor returning a null date.
- **Date** (long serialNumber)
  Constructor taking a serial number as given by Applix or Excel.
- **Date** (**Day** d, **Month** m, **Year** y)
  More traditional constructor.

inspectors

- **Weekday weekday** () const
- **Day dayOfMonth** () const
- bool **isEndOfMonth** () const
- **Day lastDayOfMonth** () const
- **Day dayOfYear** () const
  One-based (Jan 1st = 1).
- **Month month** () const
- **Year year** () const
- long **serialNumber** () const

date algebra

- **Date** & **operator+=** (int days)
  increments date in place
- **Date** & **operator-=** (int days)
  decrement date in place
- **Date** & **operator++** ()
  1-day pre-increment
- **Date operator++** (int)
  1-day post-increment

- **Date** & **operator–** ()
  1-day pre-decrement
- **Date operator–** (int)
  1-day post-decrement
- **Date operator+** (int days) const
  returns a new incremented date
- **Date operator-** (int days) const
  returns a new decremented date

other methods to increment/decrement dates

- **Date plusDays** (int days) const
- **Date plusWeeks** (int weeks) const
- **Date plusMonths** (int months) const
- **Date plusYears** (int years) const
- **Date plus** (int units, **TimeUnit**) const
- **Date plus** (const **Period** &) const

Static Public Member Functions

static methods

- bool **isLeap** (**Year** y)
- **Date minDate** ()
  earliest allowed date
- **Date maxDate** ()
  latest allowed date
- **Date todaysDate** ()
  today's date.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &, const **Date** &)
- long **operator-** (const **Date** &, const **Date** &)
  Difference in days between dates.
- bool **operator==** (const **Date** &, const **Date** &)
- bool **operator!=** (const **Date** &, const **Date** &)
- bool **operator**< (const **Date** &, const **Date** &)
- bool **operator**<= (const **Date** &, const **Date** &)
- bool **operator**> (const **Date** &, const **Date** &)
- bool **operator**>= (const **Date** &, const **Date** &)

The documentation for this class was generated from the following files:

- ql/**date.hpp**
- ql/**dataformatters.hpp**
- ql/**date.cpp**

11.114 DateFormatter Class Reference

`#include <dataformatters.hpp>`

11.114.1 Detailed Description

Formats dates for output.

Formatting can be in short (mm/dd/yyyy) or long (Month ddth, yyyy) form.

Public Types

- enum **Format** { **Long**, **Short**, **ISO** }

Static Public Member Functions

- std::string **toString** (const **Date** &d, Format f=Long)

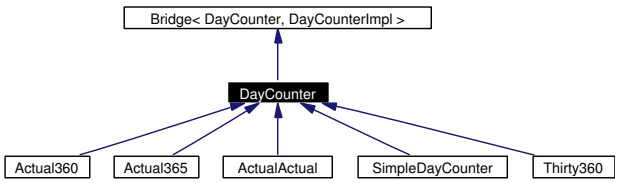
The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.115 DayCounter Class Reference

`#include <daycounter.hpp>`

Inheritance diagram for DayCounter:



11.115.1 Detailed Description

day counter class

This class provides methods for determining the length of a time period according to given market convention, both as a number of days and as a year fraction.

The Bridge pattern is used to provide the base behavior of the day counter.

Public Member Functions

- **DayCounter** ()

DayCounter interface

- std::string **name** () const
  Returns the name of the day counter.
- int **dayCount** (const **Date** &, const **Date** &) const
  Returns the number of days between two dates.
- **Time yearFraction** (const **Date** &, const **Date** &, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**()) const
  Returns the period between two dates as a fraction of year.

Protected Member Functions

- **DayCounter** (const **Handle**< **DayCounterImpl** > &impl)

Related Functions

(Note that these are not member functions.)

- bool **operator==** (const **DayCounter** &, const **DayCounter** &)
- bool **operator!=** (const **DayCounter** &, const **DayCounter** &)

11.115.2 Constructor & Destructor Documentation

11.115.2.1 DayCounter ()

This default constructor returns a day counter with a null implementation, which is therefore unusable except as a placeholder.

11.115.2.2 DayCounter (const Handle< DayCounterImpl > & *impl*) [protected]

This protected constructor will only be invoked by derived classes which define a given **DayCounter**(p. 292) implementation

11.115.3 Member Function Documentation

11.115.3.1 std::string name () const

Returns the name of the day counter.

Warning:
This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

11.115.4 Friends And Related Function Documentation

11.115.4.1 bool operator== (const DayCounter &, const DayCounter &) [related]

Returns `true` iff the two day counters belong to the same derived class.

The documentation for this class was generated from the following file:

- ql/**daycounter.hpp**

11.116 DayCounterImpl Class Reference

`#include <daycounter.hpp>`

11.116.1 Detailed Description

abstract base class for day counter implementations

Public Member Functions

- virtual std::string **name** () const =0
- virtual int **dayCount** (const **Date** &, const **Date** &) const =0
- virtual **Time yearFraction** (const **Date** &, const **Date** &, const **Date** &refPeriodStart, const **Date** &refPeriodEnd) const =0

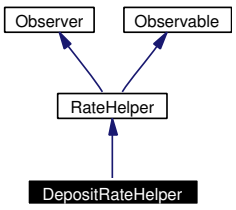
The documentation for this class was generated from the following file:

- ql/**daycounter.hpp**

11.117 DepositRateHelper Class Reference

`#include <ratehelpers.hpp>`

Inheritance diagram for DepositRateHelper:



11.117.1 Detailed Description

Deposit rate.

Warning:
This class assumes that the reference date does not change between calls of **setTermStructure()**(p. 295).

Public Member Functions

- **DepositRateHelper** (const **RelinkableHandle**< **MarketElement** > &rate, int n, **TimeUnit** units, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- **DepositRateHelper** (double rate, int n, **TimeUnit** units, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- **DiscountFactor discountGuess** () const
- void **setTermStructure** (**TermStructure** *)

  sets the term structure to be used for pricing

- **Date maturity** () const

  maturity date

11.117.2 Member Function Documentation

11.117.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:
Being a pointer and not a **Handle**(p. 390), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. 578).

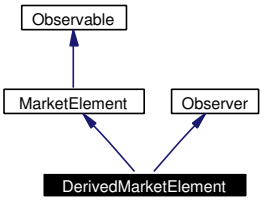
The documentation for this class was generated from the following files:

- ql/TermStructures/**ratehelpers.hpp**
- ql/TermStructures/**ratehelpers.cpp**

11.118 DerivedMarketElement Class Template Reference

`#include <marketelement.hpp>`

Inheritance diagram for DerivedMarketElement:



11.118.1 Detailed Description

template<class UnaryFunction> class QuantLib::DerivedMarketElement< UnaryFunction >

market element whose value depends on another market element

Public Member Functions

- **DerivedMarketElement** (const **RelinkableHandle**< **MarketElement** > &element, const UnaryFunction &f)

Market element interface

- double **value** () const
  returns the current value

Observer interface

- void **update** ()

11.118.2 Member Function Documentation

11.118.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

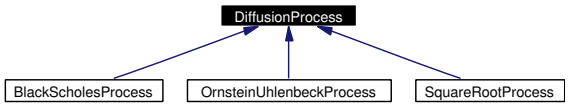
The documentation for this class was generated from the following file:

- ql/**marketelement.hpp**

11.119 DiffusionProcess Class Reference

`#include <diffusionprocess.hpp>`

Inheritance diagram for DiffusionProcess:



11.119.1 Detailed Description

Diffusion process class.

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t)dW_t.$$

Public Member Functions

- **DiffusionProcess** (double x0)
- double **x0** () const
- virtual double **drift** (**Time** t, double x) const =0
  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- virtual double **diffusion** (**Time** t, double x) const =0
  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual double **expectation** (**Time** t0, double x0, **Time** dt) const
  returns the expectation of the process after a time interval
- virtual double **variance** (**Time** t0, double x0, **Time** dt) const
  returns the variance of the process after a time interval

11.119.2 Member Function Documentation

11.119.2.1 virtual double expectation (Time *t0*, double *x0*, Time *dt*) const [virtual]

returns the expectation of the process after a time interval

returns $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $x_0 + \mu(t_0, x_0)\Delta t$.

Reimplemented in **OrnsteinUhlenbeckProcess** (p. 534).

11.119.2.2 virtual double variance (Time *t0*, double *x0*, Time *dt*) const [virtual]

returns the variance of the process after a time interval

returns $Var(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $\sigma(t_0, x_0)^2\Delta t$.

Reimplemented in **OrnsteinUhlenbeckProcess** (p. 534).

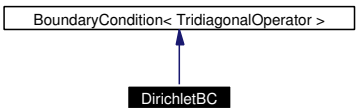
The documentation for this class was generated from the following file:

- ql/**diffusionprocess.hpp**

11.120 DirichletBC Class Reference

`#include <boundarycondition.hpp>`

Inheritance diagram for DirichletBC:



11.120.1 Detailed Description

Neumann boundary condition (i.e., constant value).

Todo
generalize to time-dependent conditions.

Public Member Functions

- **DirichletBC** (double value, Side side)
- void **applyBeforeApplying** (**TridiagonalOperator** &) const
- void **applyAfterApplying** (**Array** &) const
- void **applyBeforeSolving** (**TridiagonalOperator** &, **Array** &rhs) const
- void **applyAfterSolving** (**Array** &) const
- void **setTime** (**Time** t)

11.120.2 Member Function Documentation

11.120.2.1 void setTime (Time *t*) [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements **BoundaryCondition**< **TridiagonalOperator** > (p. 225).

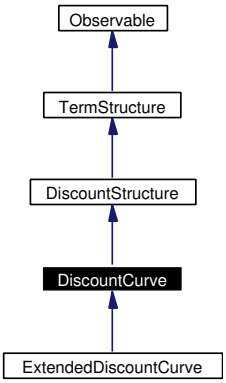
The documentation for this class was generated from the following files:

- ql/FiniteDifferences/**boundarycondition.hpp**
- ql/FiniteDifferences/**boundarycondition.cpp**

11.121 DiscountCurve Class Reference

`#include <discountcurve.hpp>`

Inheritance diagram for DiscountCurve:



11.121.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuos compounding.

Public Member Functions

- **DiscountCurve** (const **Date** &todaysDate, const std::vector< **Date** > &dates, const std::vector< **DiscountFactor** > &dfs, const **DayCounter** &dayCounter=**DayCounters::Actual365**())
- **Date todaysDate** () const

  today's date

- **Date referenceDate** () const

  the reference date, i.e., the date at which discount = 1

- **DayCounter dayCounter** () const

  the day counter used for date/time conversion

- **Date maxDate** () const

  the latest date for which the curve can return rates

- **Time maxTime** () const

  the latest time for which the curve can return rates

- const std::vector< **Time** > & **times** () const
- const std::vector< **Date** > & **dates** () const
- const std::vector< **DiscountFactor** > & **discounts** () const

Protected Types

- typedef **Math::LogLinearInterpolation**< std::vector< **Time** >::const_iterator, std::vector< **DiscountFactor** >::const_iterator > **DfInterpolation**

Protected Member Functions

- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const

  discount calculation

- int **referenceNode** (**Time**, bool extrapolate=false) const

Protected Attributes

- **Date todaysDate_**
- **Date referenceDate_**
- **DayCounter dayCounter_**
- std::vector< **Date** > **dates_**
- std::vector< **DiscountFactor** > **discounts_**
- std::vector< **Time** > **times_**
- **Handle**< **DfInterpolation** > **interpolation_**

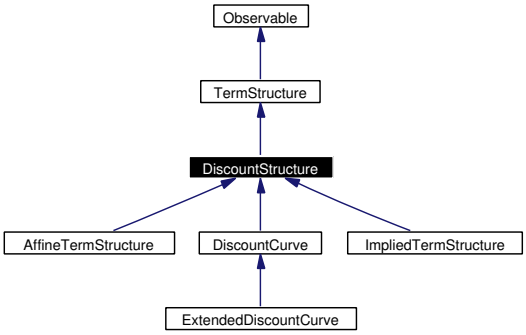
The documentation for this class was generated from the following files:

- ql/TermStructures/**discountcurve.hpp**
- ql/TermStructures/**discountcurve.cpp**

11.122 DiscountStructure Class Reference

`#include <termstructure.hpp>`

Inheritance diagram for DiscountStructure:



11.122.1 Detailed Description

Discount factor term structure.

This abstract class acts as an adapter to **TermStructure**(p. 634) allowing the programmer to implement only the `discountImpl(const Date&, bool)` method in derived classes.

Rates are assumed to be annual continuos compounding.

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const
- **Rate forwardImpl** (**Time**, bool extrapolate=false) const
- **Rate compoundForwardImpl** (**Time**, int, bool extrapolate=false) const

11.122.2 Member Function Documentation

11.122.2.1 Rate zeroYieldImpl (Time, bool *extrapolate* = false) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the discount.

Implements **TermStructure** (p. 635).

11.122.2.2 Rate forwardImpl (Time, bool *extrapolate* = false) const [protected, virtual]

Returns the instantaneous forward rate for the given date calculating it from the discount.

Implements **TermStructure** (p. 635).

11.122.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 636).

Reimplemented in **ExtendedDiscountCurve** (p. 341).

The documentation for this class was generated from the following file:

- ql/**termstructure.hpp**

11.123 DiscrepancyStatistics Class Reference

`#include <discrepancystatistics.hpp>`

Inheritance diagram for DiscrepancyStatistics:

11.123.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from **SequenceStatistics**<**Statistics**>(p. 591) and adds L^2 discrepancy calculation

Public Member Functions

- **DiscrepancyStatistics** (**Size** dimension)
- template<class Sequence> void **add** (const Sequence &sample, double weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, double weight=1.0)
- void **reset** (**Size** dimension=0)

1-dimensional inspectors

- double **discrepancy** () const

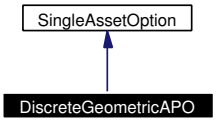
The documentation for this class was generated from the following files:

- ql/Math/**discrepancystatistics.hpp**
- ql/Math/**discrepancystatistics.cpp**

11.124 DiscreteGeometricAPO Class Reference

`#include <discretegeometricapo.hpp>`

Inheritance diagram for DiscreteGeometricAPO:



11.124.1 Detailed Description

Discrete Geometric Average Price Asian **Option**(p. 533) (European style).

This class implements a discrete geometric average price asian option, with european exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag65-97

Todo

add analytical greeks

Public Member Functions

- **DiscreteGeometricAPO** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, const std::vector< **Time** > &times, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- **Handle**< **SingleAssetOption** > **clone** () const

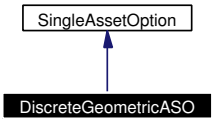
The documentation for this class was generated from the following files:

- ql/Pricers/**discretegeometricapo.hpp**
- ql/Pricers/**discretegeometricapo.cpp**

11.125 DiscreteGeometricASO Class Reference

`#include <discretegeometricaso.hpp>`

Inheritance diagram for DiscreteGeometricASO:



11.125.1 Detailed Description

Discrete Geometric Average Strike Asian **Option**(p. 533) (European style).

This class implements a discrete geometric average strike asian option, with european exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag65-97

Todo

add analytical greeks

Public Member Functions

- **DiscreteGeometricASO** (Option::Type type, double underlying, **Spread** dividendYield, **Rate** riskFreeRate, const std::vector< **Time** > &times, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- **Handle**< **SingleAssetOption** > **clone** () const

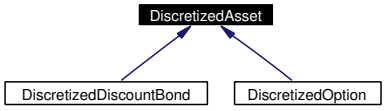
The documentation for this class was generated from the following files:

- ql/Pricers/**discretegeometricaso.hpp**
- ql/Pricers/**discretegeometricaso.cpp**

11.126 DiscretizedAsset Class Reference

`#include <discretizedasset.hpp>`

Inheritance diagram for DiscretizedAsset:



11.126.1 Detailed Description

Discretized asset class used by numerical methods.

Public Member Functions

- **DiscretizedAsset** (const **Handle**< **NumericalMethod** > &method)
- virtual void **reset** (**Size** size)=0
- **Time time** () const
- **Time** & **time** ()
- const **Array** & **values** () const
- **Array** & **values** ()
- const **Handle**< **NumericalMethod** > & **method** () const
- virtual void **preAdjustValues** ()
- virtual void **postAdjustValues** ()
- void **adjustValues** ()
- virtual void **addTimesTo** (std::list< **Time** > &times) const

Protected Member Functions

- bool **isOnTime** (**Time** t) const

Protected Attributes

- **Time time_**
- **Array values_**

11.126.2 Member Function Documentation

11.126.2.1 virtual void preAdjustValues () [virtual]

This method will be invoked after rollback and before any other asset (i.e., an option on this one) has any chance to look at the values. For instance, payments happening at times already spanned by the rollback will be added here.

11.126.2.2 virtual void postAdjustValues () [virtual]

This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

Reimplemented in **DiscretizedOption** (p. 311).

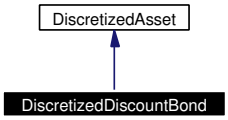
The documentation for this class was generated from the following file:

- ql/**discretizedasset.hpp**

11.127 DiscretizedDiscountBond Class Reference

`#include <discretizedasset.hpp>`

Inheritance diagram for DiscretizedDiscountBond:



11.127.1 Detailed Description

Useful discretized discount bond asset.

Public Member Functions

- **DiscretizedDiscountBond** (const **Handle**< **NumericalMethod** > &method)
- void **reset** (**Size** size)

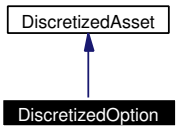
The documentation for this class was generated from the following file:

- ql/**discretizedasset.hpp**

11.128 DiscretizedOption Class Reference

`#include <discretizedasset.hpp>`

Inheritance diagram for DiscretizedOption:



11.128.1 Detailed Description

Discretized option on another asset.

Precondition:
The underlying asset must be initialized

Public Member Functions

- **DiscretizedOption** (const **Handle**< **DiscretizedAsset** > &underlying, Exercise::Type exerciseType, const std::vector< **Time** > &exerciseTimes)
- void **reset** (**Size** size)
- void **postAdjustValues** ()
- void **addTimesTo** (std::list< **Time** > &times) const

Protected Member Functions

- void **applyExerciseCondition** ()

Protected Attributes

- **Handle**< **DiscretizedAsset** > **underlying_**
- Exercise::Type **exerciseType_**
- std::vector< **Time** > **exerciseTimes_**

11.128.2 Member Function Documentation

11.128.2.1 void postAdjustValues () [virtual]

This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

Reimplemented from **DiscretizedAsset** (p. 309).

The documentation for this class was generated from the following files:

- ql/**discretizedasset.hpp**
- ql/**discretizedasset.cpp**

11.129 Disposable Class Template Reference

`#include <disposable.hpp>`

11.129.1 Detailed Description

template<class T> class QuantLib::Disposable< T >

generic disposable object with move semantics

This class can be used for returning a value by copy. It relies on the returned object exposing a `swap(T&)` method through which the copy constructor and assignment operator are implemented, thus resulting in actual move semantics. Typical use of this class is along the following lines:

```
Disposable<Foo> bar(int i) {
    Foo f(i*2);
    return f;
}
```

Warning:

In order to avoid copies in code such as shown above, the conversion from `T` to **Disposable**(p. 313)<T> is destructive, i.e., it does **not** preserve the state of the original object. Therefore, it is necessary for the developer to avoid code such as

```
Disposable<Foo> bar(Foo& f) {
    return f;
}
```

which would likely render the passed object unusable. The correct way to obtain the desired behavior would be:

```
Disposable<Foo> bar(Foo& f) {
    Foo temp = f;
    return temp;
}
```

Public Member Functions

- **Disposable** (T &t)
- **Disposable** (const **Disposable**< T > &t)
- **Disposable**< T > & **operator=** (const **Disposable**< T > &t)

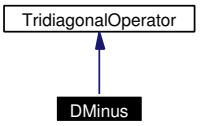
The documentation for this class was generated from the following file:

- ql/**disposable.hpp**

11.130 DMinus Class Reference

`#include <dminus.hpp>`

Inheritance diagram for DMinus:



11.130.1 Detailed Description

D_- matricial representation

The differential operator D_- discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_i - u_{i-1}}{h} = D_- u_i$$

Public Member Functions

- **DMinus** (**Size** gridPoints, double h)

The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**dminus.hpp**

11.131 DoubleFormatter Class Reference

`#include <dataformatters.hpp>`

11.131.1 Detailed Description

Formats doubles for output.

Examples:
DiscreteHedging.cpp.

Static Public Member Functions

- std::string **toString** (double x, int precision=6, int digits=0)
- std::string **toExponential** (double x, int precision=6, int digits=0)

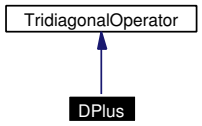
The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.132 DPlus Class Reference

`#include <dplus.hpp>`

Inheritance diagram for DPlus:



11.132.1 Detailed Description

D_+ matricial representation

The differential operator D_+ discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_i}{h} = D_+ u_i$$

Public Member Functions

- **DPlus** (**Size** gridPoints, double h)

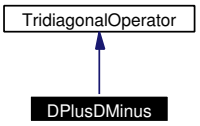
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**dplus.hpp**

11.133 DPlusDMinus Class Reference

`#include <dplusdminus.hpp>`

Inheritance diagram for DPlusDMinus:



11.133.1 Detailed Description

D_+D_- matricial representation

The differential operator D_+D_- discretizes the second derivative with the second-order formula

$$\frac{\partial^2 u_i}{\partial x^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = D_+D_-u_i$$

Public Member Functions

- **DPlusDMinus** (**Size** gridPoints, double h)

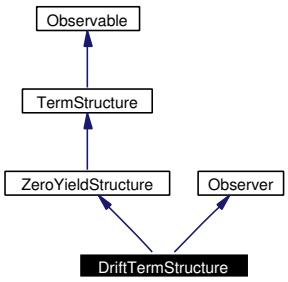
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**dplusdminus.hpp**

11.134 DriftTermStructure Class Reference

`#include <drifttermstructure.hpp>`

Inheritance diagram for DriftTermStructure:



11.134.1 Detailed Description

Drift term structure.

Drift term structure for modelling the common drift term: riskFreeRate - dividendYield - 0.5∗vol∗vol

Note:
This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **DriftTermStructure** (const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **BlackVolTermStructure** > &blackVolTS)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date todaysDate** () const
  today's date
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
  the latest date for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const

  returns the discount factor as seen from the evaluation date

11.134.2 Member Function Documentation

11.134.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

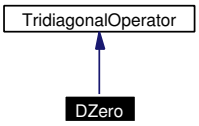
The documentation for this class was generated from the following file:

- ql/TermStructures/**drifttermstructure.hpp**

11.135 DZero Class Reference

`#include <dzero.hpp>`

Inheritance diagram for DZero:



11.135.1 Detailed Description

D_0 matricial representation

The differential operator D_0 discretizes the first derivative with the second-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_{i-1}}{2h} = D_0 u_i$$

Public Member Functions

- **DZero** (**Size** gridPoints, double h)

The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**dzero.hpp**

11.136 EndCriteria Class Reference

`#include <criteria.hpp>`

11.136.1 Detailed Description

class to gather criteria to end optimization process :

- stationary point
- stationary gradient
- maximum number of iterations ....

Public Types

- enum **Type** { **maxIter**, **statPt**, **statGd** }

Public Member Functions

- **EndCriteria** ()
  default constructor
- **EndCriteria** (int maxIteration, double epsilon)
  initialization constructor
- void **setPositiveOptimization** ()
- bool **checkIterationNumber** (int iteration)
- bool **checkStationaryValue** (double fold, double fnew)
- bool **checkAccuracyValue** (double f)
- bool **checkStationaryGradientNorm** (double normDiff)
- bool **checkAccuracyGradientNorm** (double norm)
- bool **operator()** (int iteration, double fold, double normgold, double fnew, double normgnew, double normdiff)
  test if the number of iteration is not too big and if we don't
- int **criteria** () const
  return the end criteria type

Protected Attributes

- int **maxIteration_**
  Maximum number of iterations.
- double **functionEpsilon_**
  function and gradient epsilons
- double **gradientEpsilon_**

function and gradient epsilons

- int **maxIterStatPt_**

  Maximun number of iterations in stationary state.

- int **statState_**

  Maximun number of iterations in stationary state.

- int **endCriteria_**
- bool **positiveOptimization_**

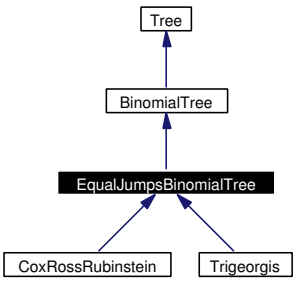
The documentation for this class was generated from the following file:

- ql/Optimization/**criteria.hpp**

11.137 EqualJumpsBinomialTree Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for EqualJumpsBinomialTree:



11.137.1 Detailed Description

base class for equal jumps binomial tree

Public Member Functions

- **EqualJumpsBinomialTree** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)
- double **underlying** (**Size** i, **Size** index) const
- double **probability** (**Size**, **Size**, **Size** branch) const

Protected Attributes

- double **dx_**
- double **pu_**
- double **pd_**

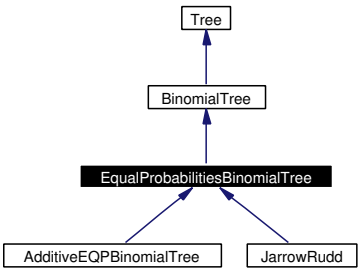
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.138 EqualProbabilitiesBinomialTree Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for EqualProbabilitiesBinomialTree:



11.138.1 Detailed Description

base class for equal probabilities binomial tree

Public Member Functions

- **EqualProbabilitiesBinomialTree** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)
- double **underlying** (**Size** i, **Size** index) const
- double **probability** (**Size**, **Size**, **Size**) const

Protected Attributes

- double **up_**

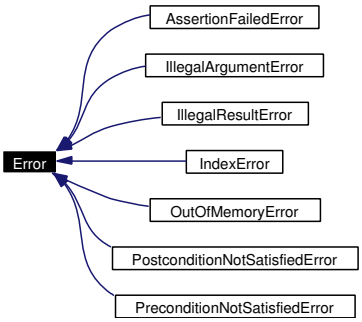
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.139 Error Class Reference

`#include <errors.hpp>`

Inheritance diagram for Error:



11.139.1 Detailed Description

Base error class.

Public Member Functions

- **Error** (const std::string &what="")
- const char * **what** () const throw ()

  returns the error message.

Static Public Member Functions

- std::string **where** (const char *file, int line)

The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.140 ErrorFunction Class Reference

`#include <errorfunction.hpp>`

11.140.1 Detailed Description

Error(p. 325) function.

formula here ... Used to calculate the cumulative normal distribution function

Public Member Functions

- double **operator()** (double x) const

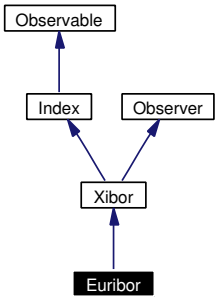
The documentation for this class was generated from the following files:

- ql/Math/**errorfunction.hpp**
- ql/Math/**errorfunction.cpp**

11.141 Euribor Class Reference

`#include <euribor.hpp>`

Inheritance diagram for Euribor:



11.141.1 Detailed Description

Euribor index

Public Member Functions

- **Euribor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual360**())

The documentation for this class was generated from the following file:

- ql/Indexes/**euribor.hpp**

11.142 EuroFormatter Class Reference

`#include <dataformatters.hpp>`

11.142.1 Detailed Description

Formats amounts in Euro for output.

Formatting follows Euro convention (x,xxx,xxx.xx)

Static Public Member Functions

- std::string **toString** (double amount)

The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.143 EuropeanExercise Class Reference

`#include <exercise.hpp>`

Inheritance diagram for EuropeanExercise:

11.143.1 Detailed Description

European exercise class.

A European option can only be exercised at one date.

Public Member Functions

- **EuropeanExercise** (**Date** date)

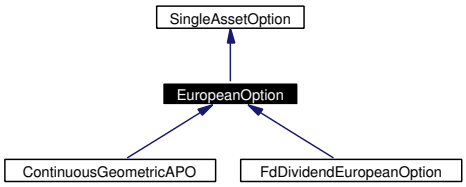
The documentation for this class was generated from the following files:

- ql/**exercise.hpp**
- ql/**exercise.cpp**

11.144 EuropeanOption Class Reference

`#include <europeanoption.hpp>`

Inheritance diagram for EuropeanOption:



11.144.1 Detailed Description

Black-Scholes-Merton European option.

Deprecated
use VanillaOption with EuropeanAnalyticEngine

Examples:
DiscreteHedging.cpp.

Public Member Functions

- **EuropeanOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- **Handle**< **SingleAssetOption** > **clone** () const
- void **setVolatility** (double newVolatility)
- void **setRiskFreeRate** (**Rate** newRate)
- void **setDividendYield** (**Rate** newDividendYield)
- double **beta** () const

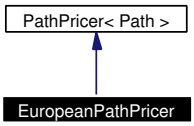
The documentation for this class was generated from the following files:

- ql/Pricers/**europeanoption.hpp**
- ql/Pricers/**europeanoption.cpp**

11.145 EuropeanPathPricer Class Reference

`#include <europeanpathpricer.hpp>`

Inheritance diagram for EuropeanPathPricer:



11.145.1 Detailed Description

path pricer for European options

Public Member Functions

- **EuropeanPathPricer** (Option::Type type, double underlying, double strike, const **RelinkableHandle**< **TermStructure** > &riskFreeTS)
- double **operator()** (const **Path** &path) const

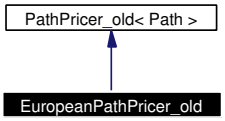
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**europeanpathpricer.hpp**
- ql/MonteCarlo/**europeanpathpricer.cpp**

11.146 EuropeanPathPricer_old Class Reference

`#include <europeanpathpricer.hpp>`

Inheritance diagram for EuropeanPathPricer_old:



11.146.1 Detailed Description

path pricer for European options

Public Member Functions

- **EuropeanPathPricer_old** (Option::Type type, double underlying, double strike, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

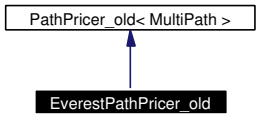
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**europeanpathpricer.hpp**
- ql/MonteCarlo/**europeanpathpricer.cpp**

11.147 EverestPathPricer_old Class Reference

`#include <everestpathpricer.hpp>`

Inheritance diagram for EverestPathPricer_old:



11.147.1 Detailed Description

path pricer for European-type Everest option

The payoff of an Everest option is given by the final-price initial-price ratio of the worst performer.

Public Member Functions

- **EverestPathPricer_old** (**DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **MultiPath** &multiPath) const

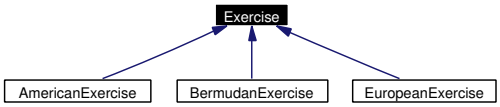
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**everestpathpricer.hpp**
- ql/MonteCarlo/**everestpathpricer.cpp**

11.148 Exercise Class Reference

`#include <exercise.hpp>`

Inheritance diagram for Exercise:



11.148.1 Detailed Description

Exercise(p. 334) class (American, Bermudan or European).

Warning:
the input dates must be effective (adjusted) exercise dates.

Public Types

- enum **Type** { **American**, **Bermudan**, **European** }

Public Member Functions

- bool **isNull** () const
- Type **type** () const
- **Date date** (**Size** index) const
- const std::vector< **Date** > & **dates** () const
- **Date lastDate** () const

Protected Attributes

- std::vector< **Date** > **dates_**
- Type **type_**

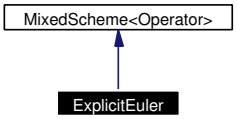
The documentation for this class was generated from the following file:

- ql/**exercise.hpp**

11.149 ExplicitEuler Class Template Reference

`#include <expliciteuler.hpp>`

Inheritance diagram for ExplicitEuler:



11.149.1 Detailed Description

template<class Operator> class QuantLib::FiniteDifferences::ExplicitEuler< Operator >

Forward Euler scheme for finite difference methods.

See sect. **The finite differences framework**(p. 31) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator-(const Operator&, const Operator&);
```

Todo

add Richardson extrapolation

Friends

- class **FiniteDifferenceModel< ExplicitEuler< Operator > >**

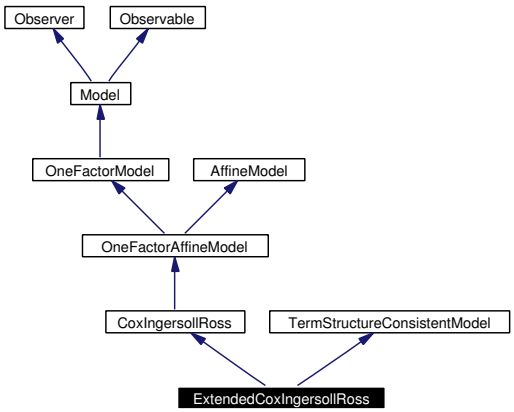
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**expliciteuler.hpp**

11.150 ExtendedCoxIngersollRoss Class Reference

`#include <extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss:



11.150.1 Detailed Description

Extended Cox-Ingersoll-Ross model class.

This class implements the extended Cox-Ingersoll-Ross model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sqrt{r_t}\sigma dW_t.$$

Unstable
This class was not tested enough to guarantee its functionality.

Public Member Functions

- **ExtendedCoxIngersollRoss** (const **RelinkableHandle**< **TermStructure** > &termStructure, double theta=0.1, double k=0.1, double sigma=0.1, double x0=0.05)
- **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Return by default a trinomial recombining tree.

- **Handle**< ShortRateDynamics > **dynamics** () const

  returns the short-rate dynamics

- double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const

Protected Member Functions

- void **generateArguments** ()
- double **A** (**Time** t, **Time** T) const

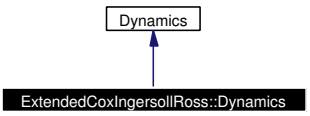
The documentation for this class was generated from the following files:

- ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.hpp**
- ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.cpp**

11.151 ExtendedCoxIngersollRoss::Dynamics Class Reference

`#include <extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:



11.151.1 Detailed Description

Short-rate dynamics in the extended Cox-Ingersoll-Ross model.

The short-rate is here

$$r_t = \varphi(t) + y_t^2$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and y_t is the state variable, the square-root of a standard CIR process.

Public Member Functions

- **Dynamics** (const **Parameter** &phi, double theta, double k, double sigma, double x0)
- virtual double **variable** (**Time** t, **Rate** r) const
- virtual double **shortRate** (**Time** t, double y) const

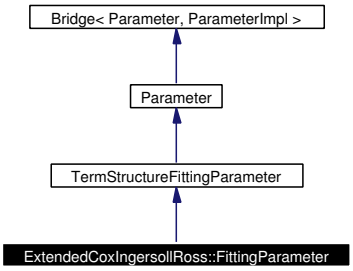
The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.hpp**

11.152 ExtendedCoxIngersollRoss::FittingParameter Class Reference

`#include <extendedcoxingersollross.hpp>`

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



11.152.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) - \frac{2k\theta(e^{th}-1)}{2h+(k+h)(e^{th}-1)} - \frac{4x_0h^2e^{th}}{(2h+(k+h)(e^{th}-1))^1},$$

where $f(t)$ is the instantaneous forward rate at t and $h = \sqrt{k^2+2\sigma^2}$.

Public Member Functions

- **FittingParameter** (const **RelinkableHandle**< **TermStructure** > &termStructure, double theta, double k, double sigma, double x0)

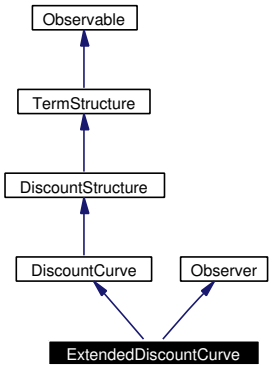
The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**extendedcoxingersollross.hpp**

11.153 ExtendedDiscountCurve Class Reference

`#include <extendeddiscountcurve.hpp>`

Inheritance diagram for ExtendedDiscountCurve:



11.153.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuos compounding.

Public Member Functions

- **ExtendedDiscountCurve** (const **Date** &todaysDate, const std::vector< **Date** > &dates, const std::vector< **DiscountFactor** > &dfs, const **Calendar** &calendar, const **RollingConvention** roll, const **DayCounter** &dayCounter=**DayCounters::Actual365**())
- **Calendar calendar** () const
- **RollingConvention roll** () const

Observer interface

- void **update** ()

Protected Member Functions

- void **calibrateNodes** () const
- **Handle**< **TermStructure** > **reversebootstrap** (int) const
- **Rate compoundForwardImpl** (**Time**, int, bool extrapolate=false) const
- **Handle**< **TermStructure** > **forwardCurve** (int) const

11.153.2 Member Function Documentation

11.153.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.153.2.2 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Reimplemented from **DiscountStructure** (p. 304).

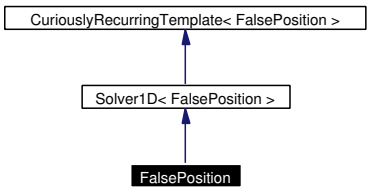
The documentation for this class was generated from the following files:

- ql/TermStructures/**extendeddiscountcurve.hpp**
- ql/TermStructures/**extendeddiscountcurve.cpp**

11.154 FalsePosition Class Reference

`#include <falseposition.hpp>`

Inheritance diagram for FalsePosition:



11.154.1 Detailed Description

False position 1-D solver.

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

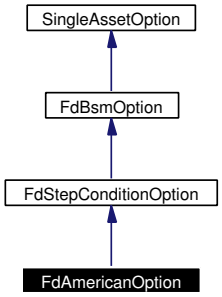
The documentation for this class was generated from the following file:

- ql/Solvers1D/**falseposition.hpp**

11.155 FdAmericanOption Class Reference

`#include <fdamericanoption.hpp>`

Inheritance diagram for FdAmericanOption:



11.155.1 Detailed Description

American option

Todo
make american call with no dividends = european

Public Member Functions

- **FdAmericanOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, int timeSteps, int gridPoints)
- void **initializeStepCondition** () const
- **Handle**< **SingleAssetOption** > **clone** () const

The documentation for this class was generated from the following file:

- ql/Pricers/**fdamericanoption.hpp**

11.156 FdBermudanOption Class Reference

`#include <fdbermudanoption.hpp>`

11.156.1 Detailed Description

Bermudan option.

Public Member Functions

- **FdBermudanOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, const std::vector< **Time** > &dates=std::vector< **Time** >(), int timeSteps=100, int gridPoints=100)
- **Handle**< **SingleAssetOption** > **clone** () const

Protected Member Functions

- void **initializeStepCondition** () const
- void **executeIntermediateStep** (int) const

Protected Attributes

- double **extraTermInBermudan**

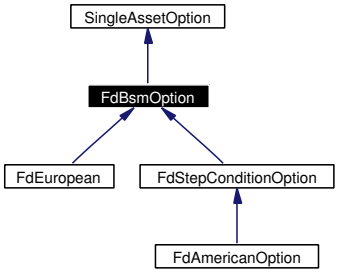
The documentation for this class was generated from the following files:

- ql/Pricers/**fdbermudanoption.hpp**
- ql/Pricers/**fdbermudanoption.cpp**

11.157 FdBsmOption Class Reference

`#include <fdbsmoption.hpp>`

Inheritance diagram for FdBsmOption:



11.157.1 Detailed Description

Black-Scholes-Merton option priced numerically.

Public Member Functions

- **FdBsmOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, **Size** gridPoints)
- virtual void **calculate** () const =0
- double **value** () const
- double **delta** () const
- double **gamma** () const
- const **Array** & **getGrid** () const

Protected Types

- typedef **FiniteDifferences::BoundaryCondition**< **FiniteDifferences::TridiagonalOperator** > **BoundaryCondition**

Protected Member Functions

- virtual void **setGridLimits** (double center, double timeDelay) const
- virtual void **initializeGrid** () const
- virtual void **initializeInitialCondition** () const
- virtual void **initializeOperator** () const

Protected Attributes

- **Size gridPoints_**
- double **value_**
- double **delta_**

- double **gamma_**
- **Array grid_**
- **FiniteDifferences::BSMOperator finiteDifferenceOperator_**
- **Array intrinsicValues_**
- std::vector< **Handle**< **BoundaryCondition** > > **BCs_**
- double **sMin_**
- double **center_**
- double **sMax_**

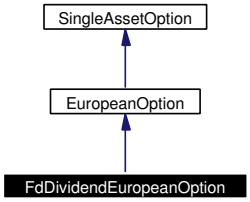
The documentation for this class was generated from the following files:

- ql/Pricers/**fdbsmoption.hpp**
- ql/Pricers/**fdbsmoption.cpp**

11.158 FdDividendEuropeanOption Class Reference

`#include <fddividendeuropeanoption.hpp>`

Inheritance diagram for FdDividendEuropeanOption:



11.158.1 Detailed Description

European option with dividends.

Public Member Functions

- **FdDividendEuropeanOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, const std::vector< double > &dividends, const std::vector< **Time** > &exdivdates)
- double **theta** () const
- double **rho** () const
- double **dividendRho** () const
- **Handle**< **SingleAssetOption** > **clone** () const
- double **riskless** (**Rate** r, std::vector< double > divs, std::vector< **Time** > divDates) const

The documentation for this class was generated from the following files:

- ql/Pricers/**fddividendeuropeanoption.hpp**
- ql/Pricers/**fddividendeuropeanoption.cpp**

11.159 FdDividendShoutOption Class Reference

`#include <fddividendshoutoption.hpp>`

11.159.1 Detailed Description

Shout option with dividends.

Public Member Functions

- **FdDividendShoutOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, const std::vector< double > &dividends=std::vector< double >(), const std::vector< **Time** > &exdivdates=std::vector< **Time** >(), int timeSteps=100, int gridPoints=100)
- **Handle**< **SingleAssetOption** > **clone** () const
- double **dividendRho** () const

Protected Member Functions

- void **initializeStepCondition** () const

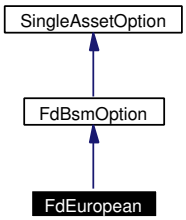
The documentation for this class was generated from the following files:

- ql/Pricers/**fddividendshoutoption.hpp**
- ql/Pricers/**fddividendshoutoption.cpp**

11.160 FdEuropean Class Reference

`#include <fdeuropean.hpp>`

Inheritance diagram for FdEuropean:



11.160.1 Detailed Description

Example of European option calculated using finite differences.

Public Member Functions

- **FdEuropean** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, **Size** timeSteps=200, **Size** gridPoints=800)
- const **Array** & **getPrices** () const
- **Handle**< **SingleAssetOption** > **clone** () const

Protected Member Functions

- void **calculate** () const

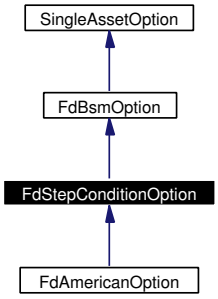
The documentation for this class was generated from the following files:

- ql/Pricers/**fdeuropean.hpp**
- ql/Pricers/**fdeuropean.cpp**

11.161 FdStepConditionOption Class Reference

`#include <fdstepconditionoption.hpp>`

Inheritance diagram for FdStepConditionOption:



11.161.1 Detailed Description

option executing additional code at each time step

Protected Member Functions

- **FdStepConditionOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility, int timeSteps, int gridPoints)
- void **calculate** () const
- virtual void **initializeStepCondition** () const =0

Protected Attributes

- **Handle**< **FiniteDifferences::StandardStepCondition** > **stepCondition_**
- int **timeSteps_**

The documentation for this class was generated from the following files:

- ql/Pricers/**fdstepconditionoption.hpp**
- ql/Pricers/**fdstepconditionoption.cpp**

11.162 filtering_iterator Class Template Reference

`#include <filteringiterator.hpp>`

11.162.1 Detailed Description

template<class Iterator, class UnaryPredicate> class QuantLib::Utilities::filtering_iterator< Iterator, UnaryPredicate >

Iterator filtering undesired data.

This iterator advances an underlying iterator returning only those data satisfying a given condition.

Public Member Functions

- **filtering_iterator** (const Iterator &, const UnaryPredicate &, const Iterator &beforeBegin, const Iterator &end)

Dereferencing

- reference **operator** * () const
- pointer **operator** → () const

Increment and decrement

- **filtering_iterator** & **operator++** ()
- **filtering_iterator operator++** (int)
- **filtering_iterator** & **operator–** ()
- **filtering_iterator operator–** (int)

Comparisons

- bool **operator==** (const **filtering_iterator**< Iterator, UnaryPredicate > &)
- bool **operator!=** (const **filtering_iterator**< Iterator, UnaryPredicate > &)

Public Attributes

- typedef< Iterator >::pointer **pointer**
- typedef< Iterator >::reference **reference**

Related Functions

(Note that these are not member functions.)

- **filtering_iterator**< Iterator, UnaryPredicate > **make_filtering_iterator** (Iterator it, UnaryPredicate p, Iterator beforeBegin, Iterator end)

  helper function to create filtering iterators

The documentation for this class was generated from the following file:

- ql/Utilities/**filteringiterator.hpp**

11.163 FiniteDifferenceModel Class Template Reference

`#include <finitedifferencemodel.hpp>`

11.163.1 Detailed Description

template<class Evolver> class QuantLib::FiniteDifferences::FiniteDifferenceModel< Evolver >

Generic finite difference model.

See sect. **The finite differences framework**(p. 31)

Public Types

- typedef Evolver::arrayType **arrayType**
- typedef Evolver::operatorType **operatorType**
- typedef **BoundaryCondition**< operatorType > **bcType**

Public Member Functions

- **FiniteDifferenceModel** (const operatorType &L, const std::vector< **Handle**< **bcType** > > &bcs, const std::vector< **Time** > &stoppingTimes=std::vector< **Time** >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< **Time** > &stoppingTimes=std::vector< **Time** >())
- void **rollback** (arrayType &a, **Time** from, **Time** to, **Size** steps, const **Handle**< **StepCondition**< arrayType > > &condition=**Handle**< **StepCondition**< arrayType > >())
- const Evolver & **evolver** () const

11.163.2 Member Function Documentation

11.163.2.1 void rollback (arrayType & *a*, Time *from*, Time *to*, Size *steps*, const Handle< StepCondition< arrayType > > & *condition* = Handle< StepCondition< arrayType > >())

solves the problem between the given times, possibly applying a condition at every step.

Warning:
being this a rollback, `from` must be a later time than `to`.

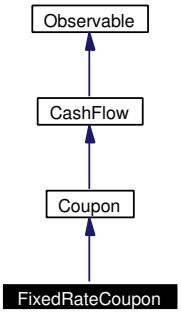
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**finitedifferencemodel.hpp**

11.164 FixedRateCoupon Class Reference

`#include <fixedratecoupon.hpp>`

Inheritance diagram for FixedRateCoupon:



11.164.1 Detailed Description

coupon paying a fixed interest rate

Public Member Functions

- **FixedRateCoupon** (double nominal, const **Date** &paymentDate, **Rate** rate, const **DayCounter** &dayCounter, const **Date** &startDate, const **Date** &endDate, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

CashFlow interface

- double **amount** () const
  returns the amount of the cash flow

Coupon interface

- **DayCounter dayCounter** () const
  day counter for accrual calculation
- double **accruedAmount** (const **Date** &) const
  accrued amount at the given date

Inspectors

- **Rate rate** () const

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.164.2 Member Function Documentation

11.164.2.1 double amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. 252).

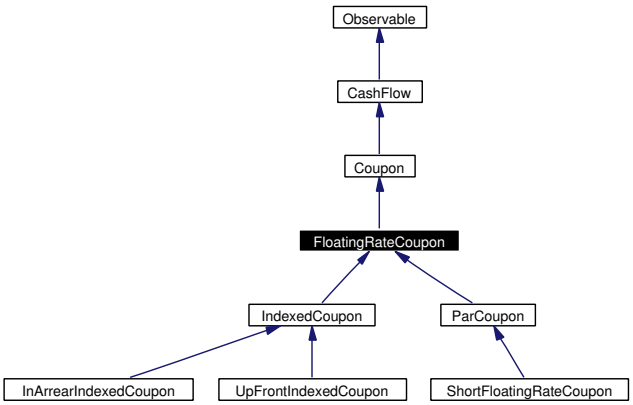
The documentation for this class was generated from the following file:

- ql/CashFlows/**fixedratecoupon.hpp**

11.165 FloatingRateCoupon Class Reference

`#include <floatingratecoupon.hpp>`

Inheritance diagram for FloatingRateCoupon:



11.165.1 Detailed Description

coupon at par on a term structure

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **FloatingRateCoupon** (double nominal, const **Date** &paymentDate, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

Coupon interface

- double **accruedAmount** (const **Date** &) const
  accrued amount at the given date

Inspectors

- int **fixingDays** () const
- virtual **Spread spread** () const
- virtual **Rate fixing** () const =0
- virtual **Date fixingDate** () const =0

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Attributes

- int **fixingDays_**
- **Spread spread_**

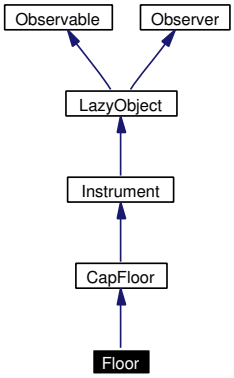
The documentation for this class was generated from the following file:

- ql/CashFlows/**floatingratecoupon.hpp**

11.166 Floor Class Reference

`#include <capfloor.hpp>`

Inheritance diagram for Floor:



11.166.1 Detailed Description

Concrete floor class.

Public Member Functions

- **Floor** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< **Rate** > &exerciseRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

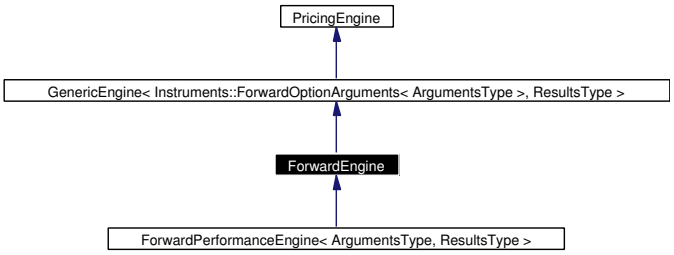
The documentation for this class was generated from the following file:

- ql/Instruments/**capfloor.hpp**

11.167 ForwardEngine Class Template Reference

`#include <forwardengines.hpp>`

Inheritance diagram for ForwardEngine:



11.167.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::PricingEngines::ForwardEngine< ArgumentsType, ResultsType >

Forward engine base class.

Public Member Functions

- **ForwardEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **setOriginalArguments** () const
- void **calculate** () const
- void **getOriginalResults** () const

Protected Attributes

- **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

The documentation for this class was generated from the following file:

- ql/PricingEngines/**forwardengines.hpp**

11.168 ForwardOptionArguments Class Template Reference

`#include <forwardvanillaoption.hpp>`

11.168.1 Detailed Description

template<class ArgumentsType> class QuantLib::Instruments::ForwardOptionArguments< ArgumentsType >

arguments for forward (strike-resetting) option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- double **moneyness**
- **Date resetDate**

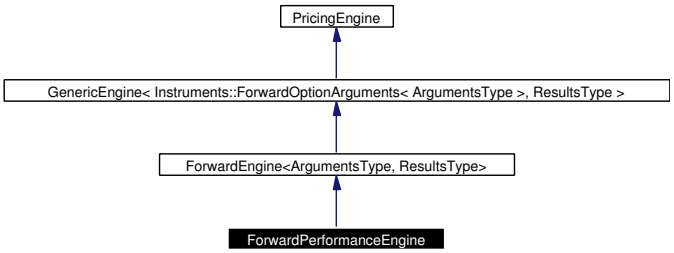
The documentation for this class was generated from the following file:

- ql/Instruments/**forwardvanillaoption.hpp**

11.169 ForwardPerformanceEngine Class Template Reference

`#include <forwardengines.hpp>`

Inheritance diagram for ForwardPerformanceEngine:



11.169.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::PricingEngines::ForwardPerformanceEngine< ArgumentsType, ResultsType >

Forward Performance engine base class.

Public Member Functions

- **ForwardPerformanceEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

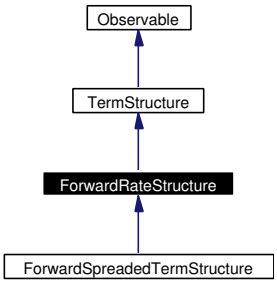
The documentation for this class was generated from the following file:

- ql/PricingEngines/**forwardengines.hpp**

11.170 ForwardRateStructure Class Reference

`#include <termstructure.hpp>`

Inheritance diagram for ForwardRateStructure:



11.170.1 Detailed Description

Forward rate term structure.

This abstract class acts as an adapter to **TermStructure**(p. 634) allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes.

Rates are assumed to be annual continuos compounding.

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const
- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const
- **Rate compoundForwardImpl** (**Time**, int, bool extrapolate=false) const

11.170.2 Member Function Documentation

11.170.2.1 Rate zeroYieldImpl (Time, bool *extrapolate* = false) const `[protected, virtual]`

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:
This is just a default, highly inefficient implementation. Derived classes should implement their own zeroYield method.

Implements **TermStructure** (p. 635).

Reimplemented in **ForwardSpreadedTermStructure** (p. 364).

11.170.2.2 DiscountFactor discountImpl (Time, bool *extrapolate* = false) const `[protected, virtual]`

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Implements **TermStructure** (p. 635).

11.170.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 636).

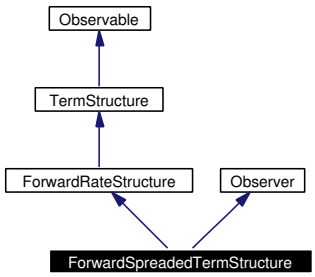
The documentation for this class was generated from the following file:

- ql/**termstructure.hpp**

11.171 ForwardSpreadedTermStructure Class Reference

`#include <forwardspreadedtermstructure.hpp>`

Inheritance diagram for ForwardSpreadedTermStructure:



11.171.1 Detailed Description

Term structure with added spread on the instantaneous forward rate.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Public Member Functions

- **ForwardSpreadedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **RelinkableHandle**< **MarketElement** > &spread)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date todaysDate** () const
  today's date
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
  the latest date for which the curve can return rates
- **Time maxTime** () const
  the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- **Rate forwardImpl** (**Time**, bool extrapolate=false) const
  returns the spreaded forward rate
- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const
  returns the spreaded zero yield rate

11.171.2 Member Function Documentation

11.171.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.171.2.2 Rate zeroYieldImpl (Time, bool *extrapolate* = false) const [protected, virtual]

returns the spreaded zero yield rate

Warning:
This method must disappear should the spread become a curve

Reimplemented from **ForwardRateStructure** (p. 361).

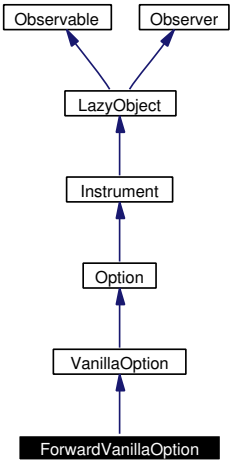
The documentation for this class was generated from the following file:

- ql/TermStructures/**forwardspreadedtermstructure.hpp**

11.172 ForwardVanillaOption Class Reference

`#include <forwardvanillaoption.hpp>`

Inheritance diagram for ForwardVanillaOption:



11.172.1 Detailed Description

Forward version of a vanilla option.

Public Types

- typedef **ForwardOptionArguments**< **VanillaOption::arguments** > **arguments**
- typedef **VanillaOption::results results**

Public Member Functions

- **ForwardVanillaOption** (Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine, double moneyness, **Date** resetDate, const std::string &isinCode="", const std::string &description="")
- void **setupArguments** (**Arguments** *) const

Protected Member Functions

- void **performCalculations** () const

11.172.2 Member Function Documentation

11.172.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **VanillaOption** (p. 665).

11.172.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **VanillaOption** (p. 666).

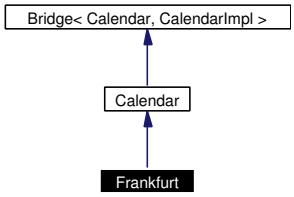
The documentation for this class was generated from the following files:

- ql/Instruments/**forwardvanillaoption.hpp**
- ql/Instruments/**forwardvanillaoption.cpp**

11.173 Frankfurt Class Reference

`#include <frankfurt.hpp>`

Inheritance diagram for Frankfurt:



11.173.1 Detailed Description

Frankfurt calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Ascension Thursday
- Whit Monday
- Corpus Christi
- Labour Day, May 1st
- National Day, October 3rd
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31st

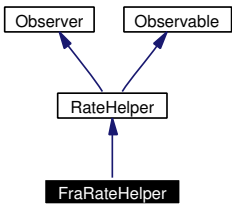
The documentation for this class was generated from the following file:

- ql/Calendars/**frankfurt.hpp**

11.174 FraRateHelper Class Reference

`#include <ratehelpers.hpp>`

Inheritance diagram for FraRateHelper:



11.174.1 Detailed Description

Forward rate agreement.

Warning:
This class assumes that the reference date does not change between calls of **setTermStructure()**(p. 368).

Todo
convexity adjustment should be implemented.

Public Member Functions

- **FraRateHelper** (const **RelinkableHandle**< **MarketElement** > &rate, int monthsToStart, int monthsToEnd, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- **FraRateHelper** (double rate, int monthsToStart, int monthsToEnd, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- **DiscountFactor discountGuess** () const
- void **setTermStructure** (**TermStructure** *)

  sets the term structure to be used for pricing

- **Date maturity** () const

  maturity date

11.174.2 Member Function Documentation

11.174.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a **Handle**(p. 390), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. 578).

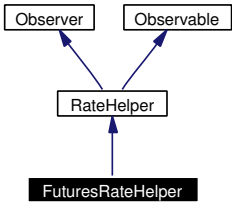
The documentation for this class was generated from the following files:

- ql/TermStructures/**ratehelpers.hpp**
- ql/TermStructures/**ratehelpers.cpp**

11.175 FuturesRateHelper Class Reference

`#include <ratehelpers.hpp>`

Inheritance diagram for FuturesRateHelper:



11.175.1 Detailed Description

Interest Rate Futures.

Warning:
This class assumes that the reference date does not change between calls of **setTermStructure()**(p. 578).

Public Member Functions

- **FuturesRateHelper** (const **RelinkableHandle**< **MarketElement** > &price, const **Date** &ImmDate, int nMonths, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- **FuturesRateHelper** (const **RelinkableHandle**< **MarketElement** > &price, const **Date** &ImmDate, const **Date** &MatDate, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- **FuturesRateHelper** (double price, const **Date** &ImmDate, int nMonths, const **Calendar** &calendar, **RollingConvention** convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- **DiscountFactor discountGuess** () const
- **Date maturity** () const

  maturity date

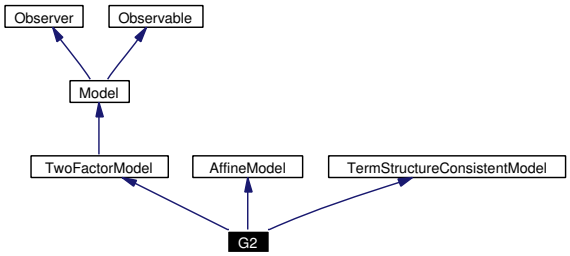
The documentation for this class was generated from the following files:

- ql/TermStructures/**ratehelpers.hpp**
- ql/TermStructures/**ratehelpers.cpp**

11.176 G2 Class Reference

`#include <g2.hpp>`

Inheritance diagram for G2:



11.176.1 Detailed Description

Two-additive-factor gaussian model class.

This class implements a two-additive-factor model defined by

$$dr_t = \varphi(t) + x_t + y_t$$

where x_t and y_t are defined by

$$dx_t = -ax_t dt + \sigma dW_t^1, x_0 = 0$$

$$dy_t = -by_t dt + \sigma dW_t^2, y_0 = 0$$

and $dW_t^1 dW_t^2 = \rho dt$.

Unstable
This class was not tested enough to guarantee its functionality.

Public Member Functions

- **G2** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.01, double b=0.1, double eta=0.01, double rho=0.9)
- **Handle**< ShortRateDynamics > **dynamics** () const
  Returns the short-rate dynamics.
- double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const
- double **swaption** (const **Instruments::Swaption::arguments** &arguments) const
- **DiscountFactor discount** (**Time** t) const
  Implied discount curve.

Protected Member Functions

- void **generateArguments** ()
- double **A** (**Time** t, **Time** T) const
- double **B** (double x, **Time** t) const

Friends

- class **SwaptionPricingFunction**

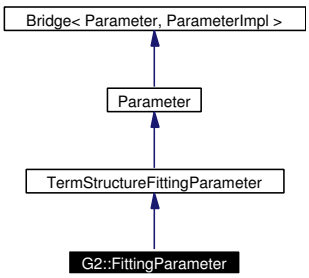
The documentation for this class was generated from the following files:

- ql/ShortRateModels/TwoFactorModels/**g2.hpp**
- ql/ShortRateModels/TwoFactorModels/**g2.cpp**

11.177 G2::FittingParameter Class Reference

`#include <g2.hpp>`

Inheritance diagram for G2::FittingParameter:



11.177.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2}\left(\frac{\sigma(1-e^{-at})}{a}\right)^2 + \frac{1}{2}\left(\frac{\eta(1-e^{-bt})}{b}\right)^2 + \rho\frac{\sigma(1-e^{-at})}{a}\frac{\eta(1-e^{-bt})}{b},$$

where $f(t)$ is the instantaneous forward rate at t.

Public Member Functions

- **FittingParameter** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a, double sigma, double b, double eta, double rho)

The documentation for this class was generated from the following file:

- ql/ShortRateModels/TwoFactorModels/**g2.hpp**

11.178 GammaFunction Class Reference

`#include <gammadistribution.hpp>`

11.178.1 Detailed Description

Gamma function class.

This is a function defined by

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

Public Member Functions

- double **logValue** (double x) const

The documentation for this class was generated from the following files:

- ql/Math/**gammadistribution.hpp**
- ql/Math/**gammadistribution.cpp**

11.179 GaussianStatistics Class Template Reference

`#include <gaussianstatistics.hpp>`

11.179.1 Detailed Description

template<class Stat> class QuantLib::Math::GaussianStatistics< Stat >

Statistics tool for gaussian-assumption risk measures.

It can calculate gaussian assumption risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the template class

Public Member Functions

Gaussian risk measures

- double **gaussianDownsideVariance** () const
- double **gaussianDownsideDeviation** () const
- double **gaussianRegret** (double target) const
- double **gaussianPercentile** (double percentile) const
- double **gaussianPotentialUpside** (double percentile) const
  gaussian-assumption Potential-Upside at a given percentile
- double **gaussianValueAtRisk** (double percentile) const
  gaussian-assumption Value-At-Risk at a given percentile
- double **gaussianExpectedShortfall** (double percentile) const
  gaussian-assumption Expected Shortfall at a given percentile
- double **gaussianShortfall** (double target) const
  gaussian-assumption Shortfall (observations below target)
- double **gaussianAverageShortfall** (double target) const
  gaussian-assumption Average Shortfall (averaged shortfallness)

11.179.2 Member Function Documentation

11.179.2.1 double gaussianDownsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^{N} \theta \times x_i^2}{\sum_{i=1}^{N} w_i}$$

, where θ = 0 if x > 0 and θ =1 if x <0

11.179.2.2 double gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

11.179.2.3 double gaussianRegret (double *target*) const

returns the variance of observations below target

$$\frac{\sum w_i(min(0, x_i - target))^2}{\sum w_i}.$$

See Dembo, Freeman "The Rules Of Risk", Wiley (2001)

11.179.2.4 double gaussianPercentile (double *percentile*) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-u^2/2)du$$

11.179.2.5 double gaussianPotentialUpside (double *percentile*) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:
percentile must be in range [90%-100%)

11.179.2.6 double gaussianValueAtRisk (double *percentile*) const

gaussian-assumption Value-At-Risk at a given percentile

Precondition:
percentile must be in range [90%-100%)

11.179.2.7 double gaussianExpectedShortfall (double *percentile*) const

gaussian-assumption Expected Shortfall at a given percentile

Precondition:
percentile must be in range 90%-100%

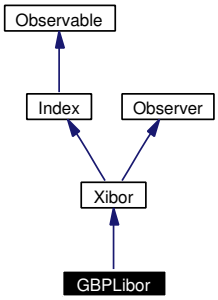
The documentation for this class was generated from the following file:

- ql/Math/**gaussianstatistics.hpp**

11.180 GBPLibor Class Reference

`#include <gbplibor.hpp>`

Inheritance diagram for GBPLibor:



11.180.1 Detailed Description

GBP Libor index

Public Member Functions

- **GBPLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual365**())

The documentation for this class was generated from the following file:

- ql/Indexes/**gbplibor.hpp**

11.181 GeneralStatistics Class Reference

`#include <generalstatistics.hpp>`

11.181.1 Detailed Description

Statistics tool.

This class accumulates a set of data and returns their statistics (e.g: mean, variance, skewness, kurtosis, error estimation, percentile, etc.) based on the empirical distribution (no gaussian assumption)

It doesn't suffer the numerical instability problem of **IncrementalStatistics**(p. 414). The downside is that it stores all samples, thus increasing the memory requirements.

Examples:
DiscreteHedging.cpp.

Public Member Functions

Inspectors

- **Size samples** () const
  number of samples collected
- const std::vector< std::pair< double, double > > & **data** () const
  collected data
- double **weightSum** () const
  sum of data weights
- double **mean** () const
- double **variance** () const
- double **standardDeviation** () const
- double **errorEstimate** () const
- double **skewness** () const
- double **kurtosis** () const
- double **min** () const
- double **max** () const
- template<class Func, class Predicate> std::pair< double, **Size** > **expectationValue** (const Func &f, const Predicate &inRange) const
- double **percentile** (double y) const
- double **topPercentile** (double y) const

Modifiers

- void **add** (double value, double weight=1.0)
  adds a datum to the set, possibly with a weight
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
  adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)

adds a sequence of data to the set, each with its weight

- void **reset** ()
  resets the data to a null set

11.181.2 Member Function Documentation

11.181.2.1 double mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

11.181.2.2 double variance () const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \left\langle (x - \langle x \rangle)^2 \right\rangle.$$

11.181.2.3 double standardDeviation () const

returns the standard deviation σ, defined as the square root of the variance.

11.181.2.4 double errorEstimate () const

returns the error estimate on the mean value, defined as $\epsilon = \sigma / \sqrt{N}$.

11.181.2.5 double skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\left\langle (x - \langle x \rangle)^3 \right\rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

11.181.2.6 double kurtosis () const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\left\langle (x - \langle x \rangle)^4 \right\rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

11.181.2.7 double min () const

returns the minimum sample value

11.181.2.8 double max () const

returns the maximum sample value

11.181.2.9 std::pair<double,Size> expectationValue (const Func & *f*, const Predicate & *inRange*) const

Expectation value of a function f on a given range $\mathcal{R}$, i.e.,

$$\mathrm{E}\left[f \mid \mathcal{R}\right] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

The range is passed as a boolean function returning `true` if the argument belongs to the range or `false` otherwise.

The function returns a pair made of the result and the number of observations in the given range.

11.181.2.10 double percentile (double *y*) const

y-th percentile, defined as the value $\bar{x}$ such that

$$y = \frac{\sum_{x_i < \bar{x}} w_i}{\sum_i w_i}$$

11.181.2.11 double topPercentile (double *y*) const

y-th top percentile, defined as the value $\bar{x}$ such that

$$y = \frac{\sum_{x_i > \bar{x}} w_i}{\sum_i w_i}$$

11.181.2.12 void add (double *value*, double *weight* = 1.0)

adds a datum to the set, possibly with a weight

Precondition:
weights must be positive or null

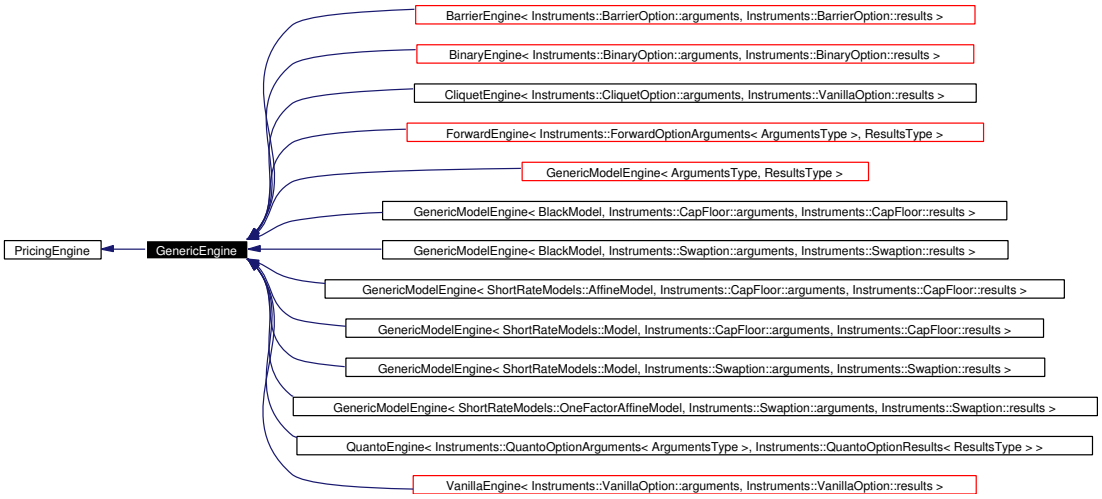
The documentation for this class was generated from the following files:

- ql/Math/**generalstatistics.hpp**
- ql/Math/**generalstatistics.cpp**

11.182 GenericEngine Class Template Reference

`#include <genericengine.hpp>`

Inheritance diagram for GenericEngine:



11.182.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::PricingEngines::GenericEngine< ArgumentsType, ResultsType >

template base class for option pricing engines

Derived engines only need to implement the `calculate()` method the inherit from **PricingEngine**(p. 559)

Public Member Functions

- **Arguments** * **arguments** () const
- const **Results** * **results** () const
- void **reset** () const

Protected Attributes

- ArgumentsType **arguments_**
- ResultsType **results_**

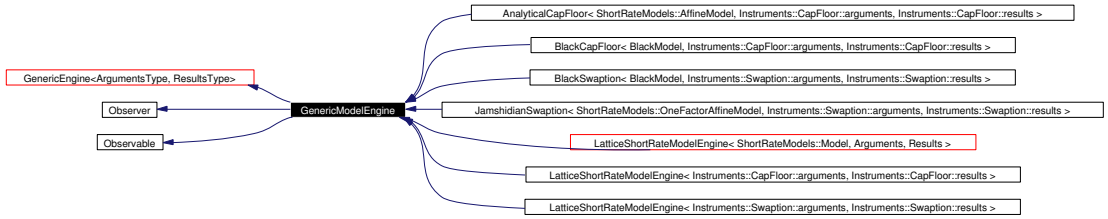
The documentation for this class was generated from the following file:

- ql/PricingEngines/**genericengine.hpp**

11.183 GenericModelEngine Class Template Reference

`#include <genericengine.hpp>`

Inheritance diagram for GenericModelEngine:



11.183.1 Detailed Description

template<class ModelType, class ArgumentsType, class ResultsType> class QuantLib::PricingEngines::GenericModelEngine< ModelType, ArgumentsType, ResultsType >

Base class for some pricing engine on a particular model.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **GenericModelEngine** (const **Handle**< ModelType > &model)
- void **validateArguments** () const
- void **setModel** (const **Handle**< ModelType > &model)
- virtual void **update** ()

Protected Attributes

- **Handle**< ModelType > **model_**

11.183.2 Member Function Documentation

11.183.2.1 virtual void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

Reimplemented in **LatticeShortRateModelEngine** (p. 442), **LatticeShortRateModelEngine**< **Instruments::Swaption::arguments, Instruments::Swaption::results** > (p. 442), and **LatticeShortRateModelEngine**< **Instruments::CapFloor::arguments, Instruments::CapFloor::results** > (p. 442).

The documentation for this class was generated from the following file:

- ql/PricingEngines/**genericengine.hpp**

11.184 GenericRiskStatistics Class Template Reference

`#include <riskstatistics.hpp>`

11.184.1 Detailed Description

template<class S> class QuantLib::Math::GenericRiskStatistics< S >

empirical-distribution risk measures

This class wraps a somewhat generic statistic tool and adds a number of risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the data distribution as reported by the underlying tool.

Public Member Functions

- double **semiVariance** () const
- double **semiDeviation** () const
- double **downsideVariance** () const
- double **downsideDeviation** () const
- double **regret** (double target) const
- double **potentialUpside** (double percentile) const

  potential upside (the reciprocal of VAR) at a given percentile

- double **valueAtRisk** (double percentile) const

  value-at-risk at a given percentile

- double **expectedShortfall** (double percentile) const

  expected shortfall at a given percentile

- double **shortfall** (double target) const
- double **averageShortfall** (double target) const

11.184.2 Member Function Documentation

11.184.2.1 double semiVariance () const

returns the variance of observations below the mean,

$$\frac{N}{N-1}\mathrm{E}\left[(x-\langle x\rangle)^2 \mid x<\langle x\rangle\right].$$

See Markowitz (1959).

11.184.2.2 double semiDeviation () const

returns the semi deviation, defined as the square root of the semi variance.

11.184.2.3 double downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1}\mathrm{E}\left[x^2 \mid x < 0\right].$$

11.184.2.4 double downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

11.184.2.5 double regret (double *target*) const

returns the variance of observations below target,

$$\frac{N}{N-1}\mathrm{E}\left[(x-t)^2 \mid x < t\right].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

11.184.2.6 double potentialUpside (double *centile*) const

potential upside (the reciprocal of VAR) at a given percentile

Precondition:
percentile must be in range [90%-100%)

11.184.2.7 double valueAtRisk (double *centile*) const

value-at-risk at a given percentile

Precondition:
percentile must be in range [90%-100%)

11.184.2.8 double expectedShortfall (double *percentile*) const

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$\mathrm{E}\left[x \mid x < \mathrm{VaR}(p)\right],$$

that is the average of observations below the given percentile p. Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

11.184.2.9 double shortfall (double *target*) const

probability of missing the given target, defined as

$$\mathrm{E}\left[\Theta \mid (-\infty, \infty)\right]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

11.184.2.10 double averageShortfall (double *target*) const

averaged shortfallness, defined as

$$\mathrm{E}\left[t - x \mid x < t\right]$$

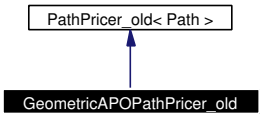
The documentation for this class was generated from the following file:

- ql/Math/riskstatistics.hpp

11.185 GeometricAPOPathPricer_old Class Reference

`#include <geometricapopathpricer.hpp>`

Inheritance diagram for GeometricAPOPathPricer_old:



11.185.1 Detailed Description

path pricer for geometric average price option

Public Member Functions

- **GeometricAPOPathPricer_old** (Option::Type type, double underlying, double strike, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

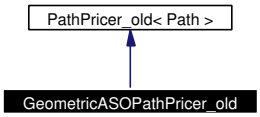
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**geometricapopathpricer.hpp**
- ql/MonteCarlo/**geometricapopathpricer.cpp**

11.186 GeometricASOPathPricer_old Class Reference

`#include <geometricasopathpricer.hpp>`

Inheritance diagram for GeometricASOPathPricer_old:



11.186.1 Detailed Description

path pricer for geometric average strike option

Public Member Functions

- **GeometricASOPathPricer_old** (Option::Type type, double underlying, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

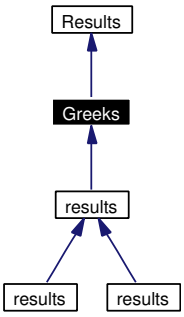
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**geometricasopathpricer.hpp**
- ql/MonteCarlo/**geometricasopathpricer.cpp**

11.187 Greeks Class Reference

`#include <option.hpp>`

Inheritance diagram for Greeks:



11.187.1 Detailed Description

additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- double **delta**
- double **gamma**
- double **theta**
- double **vega**
- double **rho**
- double **dividendRho**
- double **strikeSensitivity**

The documentation for this class was generated from the following file:

- ql/**option.hpp**

11.188 HaltonRsg Class Reference

`#include <haltonrsg.hpp>`

11.188.1 Detailed Description

Halton low-discrepancy sequence generator.

Halton algorithm for low-discrepancy sequence. For more details see chapter 8, paragraph 2 of "Monte Carlo Methods in Finance", by Peter Jäckel

Public Types

- typedef **MonteCarlo::Sample**< **Array** > **sample_type**

Public Member Functions

- **HaltonRsg** (**Size** dimensionality, unsigned long seed=0, bool randomStart=true, bool randomShift=false)
- const **sample_type** & **nextSequence** () const
- const **sample_type** & **lastSequence** () const
- **Size dimension** () const

The documentation for this class was generated from the following files:

- ql/RandomNumbers/**haltonrsg.hpp**
- ql/RandomNumbers/**haltonrsg.cpp**

11.189 Handle Class Template Reference

`#include <handle.hpp>`

11.189.1 Detailed Description

template<class T> class QuantLib::Handle< T >

Reference-counted pointer.

This class acts as a proxy to a pointer contained in it. Such pointer is owned by the handle, i.e., the handle will be responsible for its deletion, unless explicitly stated by the programmer.

A count of the references to the contained pointer is incremented every time a handle is copied, and decremented every time a handle is deleted or goes out of scope. When the handle owns the pointer, this mechanism ensures on one hand, that the pointer will not be deallocated as long as a handle refers to it, and on the other hand, that it will be deallocated when no more handles do.

Note:
The implementation of this class was originally taken from "The C++ Programming Language", 3rd ed., B.Stroustrup, Addison-Wesley, 1997.

Warning:
This mechanism will break and result in untimely deallocation of the pointer (and very possible death of your executable) if two handles are explicitly initialized with the same pointer, as in

```
SomeObj* so = new SomeObj;
Handle<SomeObj> h1(so);
Handle<SomeObj> h2 = h1;    // this is safe.
Handle<SomeObj> h3(so);     // this is definitely not.
```

It is good practice to create the pointer and immediately pass it to the handle, as in

```
Handle<SomeObj> h1(new SomeObj);    // this is as safe as can be.
```

When the programmer keeps the ownership of the pointer, as explicitly declared in

```
SomeObj so;
Handle<SomeObj> h(&so,false);
```

it is responsibility of the programmer to make sure that the object remain in scope as long as there are handles pointing to it. Also, the programmer must explicitly delete the object if required.

Examples:
DiscreteHedging.cpp.

Public Member Functions

constructors, destructor, and assignment

- **Handle** (T *ptr=0, bool owns=true)
  Constructor taking a pointer.

- template<class U> **Handle** (const **Handle**< U > &from)
- **Handle** (const **Handle** &from)
- template<class U> **Handle** & **operator=** (const **Handle**< U > &from)
- **Handle** & **operator=** (const **Handle** &from)

Dereferencing

- T & **operator** * () const
- T * **operator** → () const

Inspectors

- bool **isNull** () const

  Checks if the contained pointer is actually allocated.

- bool **operator==** (const **Handle**< T > &) const

  Checks if the two handles point to the same object.

- bool **operator!=** (const **Handle**< T > &) const

Friends

- class **HandleCopier**

11.189.2 Constructor & Destructor Documentation

11.189.2.1 Handle (T * *ptr* = 0, bool *owns* = true) [explicit]

Constructor taking a pointer.

If **owns** is set to `true` (the default), the handle will be responsible for the deletion of the pointer. If it is set to `false`, the programmer must make sure that the pointed object remains in scope for the lifetime of the handle and its copies. Destruction of the object is also responsibility of the programmer.

It is advised that handles be used with `owns = false` only in a controlled an self-contained environment. Such a case happens when an object needs to pass a handle to itself to inner classes or bootstrappers - i.e., contained or temporary objects whose lifetime is guaranteed not to last more than the lifetime of the object.

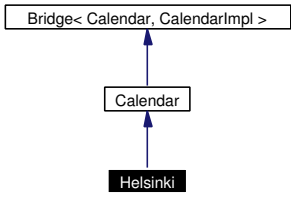
The documentation for this class was generated from the following file:

- ql/**handle.hpp**

11.190 Helsinki Class Reference

`#include <helsinki.hpp>`

Inheritance diagram for Helsinki:



11.190.1 Detailed Description

Helsinki calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension Thursday
- Labour Day, May 1st
- Midsummer Eve (Friday between June 18-24)
- Independence Day, December 6th
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th

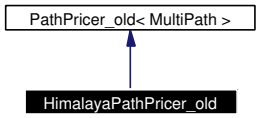
The documentation for this class was generated from the following file:

- ql/Calendars/**helsinki.hpp**

11.191 HimalayaPathPricer_old Class Reference

`#include <himalayapathpricer.hpp>`

Inheritance diagram for HimalayaPathPricer_old:



11.191.1 Detailed Description

multipath pricer for European-type Himalaya option

The payoff of an himalaya option is computed in the following way: given a basket of N assets, and M time periods, at end of each period the option which performed the best is added to the average and then discarded from the basket. At the end of the M periods the option pays the max between the strike and the average of the best performers.

Public Member Functions

- **HimalayaPathPricer_old** (const std::vector< double > &underlying, double strike, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **MultiPath** &multiPath) const

The documentation for this class was generated from the following files:

- ql/MonteCarlo/**himalayapathpricer.hpp**
- ql/MonteCarlo/**himalayapathpricer.cpp**

11.192 History Class Reference

`#include <history.hpp>`

11.192.1 Detailed Description

Container for historical data.

This class acts as a generic repository for a set of historical data. Single data can be accessed through their date, while sets of consecutive data can be accessed through iterators.

A history can contain null data, which can either be returned or skipped according to the chosen iterator type.

Example: uses of history iterators (p. **??**)

Public Types

- typedef **Utilities::filtering_iterator**< **const_iterator**, DataValidator > **const_valid_iterator**
  bidirectional iterator on non-null history entries
- typedef std::vector< double >::**const_iterator const_data_iterator**
  random access iterator on historical data
- typedef **Utilities::filtering_iterator**< **const_data_iterator**, DataValidator > **const_valid_-data_iterator**
  bidirectional iterator on non-null historical data

Public Member Functions

- **History** ()
- template<class Iterator> **History** (const **Date** &firstDate, const **Date** &lastDate, Iterator begin, Iterator end)
- **History** (const **Date** &firstDate, const std::vector< double > &values)
- **History** (const **Date** &firstDate, const **Date** &lastDate, const std::vector< double > &values)
- **History** (const std::vector< **Date** > &dates, const std::vector< double > &values)

Inspectors

- const **Date** & **firstDate** () const
  returns the first date for which a historical datum exists
- const **Date** & **lastDate** () const
  returns the last date for which a historical datum exists
- int **size** () const
  returns the number of historical data including null ones

Historical data access

- double **operator[ ]** (const **Date** &) const

  returns the (possibly null) datum corresponding to the given date

Iterator access

Four different types of iterators are provided, namely, const_iterator, const_valid_iterator, const_data_iterator, and const_valid_data_iterator.

const_iterator and const_valid_iterator point to an Entry structure, the difference being that the latter only iterates over valid entries - i.e., entries whose data are not null. The same difference exists between const_data_iterator and const_valid_data_iterator which point directly to historical values without reference to the date they are associated to.

- **const_iterator begin** () const
- **const_iterator end** () const
- **const_iterator iterator** (const **Date** &d) const
- **const_valid_iterator vbegin** () const
- **const_valid_iterator vend** () const
- **const_valid_iterator valid_iterator** (const **Date** &d) const
- **const_data_iterator dbegin** () const
- **const_data_iterator dend** () const
- **const_data_iterator data_iterator** (const **Date** &d) const
- **const_valid_data_iterator vdbegin** () const
- **const_valid_data_iterator vdend** () const
- **const_valid_data_iterator valid_data_iterator** (const **Date** &d) const

11.192.2 Constructor & Destructor Documentation

11.192.2.1 History ()

Default constructor

11.192.2.2 History (const Date & *firstDate*, const Date & *lastDate*, Iterator *begin*, Iterator *end*)

This constructor initializes the history with the given set of values, corresponding to the date range between ***firstDate*** and ***lastDate*** included.

Precondition:
begin-end must equal the number of days from ***firstDate*** to ***lastDate*** included.

11.192.2.3 History (const Date & *firstDate*, const Date & *lastDate*, const std::vector< double > & *values*)

This constructor initializes the history with the given set of values, corresponding to the date range between ***firstDate*** and ***lastDate*** included.

Precondition:
The size of ***values*** must equal the number of days from ***firstDate*** to ***lastDate*** included.

11.192.2.4 History (const std::vector< Date > & *dates*, const std::vector< double > & *values*)

This constructor initializes the history with the given set of values, corresponding each to the element with the same index in the given set of dates. The whole date range between ***dates***[0] and ***dates***[N-1] will be automatically filled by inserting null values where a date is missing from the given set.

Precondition:

dates must be sorted.

There can be no pairs (***dates***[i],***values***[i]) and (***dates***[j],***values***[j]) such that `dates[i] == dates[j] && values[i] != values[j]`. Pairs with `dates[i] == dates[j] && values[i] == values[j]` are allowed; the duplicated entries will be discarded.

The size of ***values*** must equal the number of days from ***firstDate*** to ***lastDate*** included.

The documentation for this class was generated from the following file:

- ql/**history.hpp**

11.193 History::const_iterator Class Reference

`#include <history.hpp>`

11.193.1 Detailed Description

random access iterator on history entries

Public Types

- typedef **Entry value_type**
- typedef int **difference_type**
- typedef const **Entry** * **pointer**
- typedef const **Entry** & **reference**

Public Member Functions

Dereferencing

- **reference operator** * () const
- **pointer operator** → () const

Random access

- **value_type operator[ ]** (difference_type i) const

Increment and decrement

- **const_iterator** & **operator++** ()
- **const_iterator operator++** (int)
- **const_iterator** & **operator–** ()
- **const_iterator operator–** (int)
- **const_iterator** & **operator+=** (difference_type i)
- **const_iterator** & **operator-=** (difference_type i)
- **const_iterator operator+** (difference_type i)
- **const_iterator operator-** (difference_type i)

Difference

- difference_type **operator-** (const **const_iterator** &i)

Comparisons

- bool **operator==** (const **const_iterator** &i)
- bool **operator!=** (const **const_iterator** &i)
- bool **operator**< (const **const_iterator** &i)
- bool **operator**> (const **const_iterator** &i)
- bool **operator**<= (const **const_iterator** &i)
- bool **operator**>= (const **const_iterator** &i)

Friends

- class **History**

The documentation for this class was generated from the following file:

- ql/**history.hpp**

11.194 History::Entry Class Reference

`#include <history.hpp>`

11.194.1 Detailed Description

single datum in history

Public Member Functions

- const **Date** & **date** () const
- double **value** () const

Friends

- class **const_iterator**

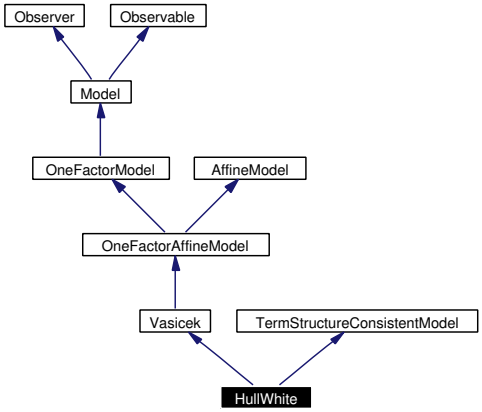
The documentation for this class was generated from the following file:

- ql/**history.hpp**

11.195 HullWhite Class Reference

`#include <hullwhite.hpp>`

Inheritance diagram for HullWhite:



11.195.1 Detailed Description

Single-factor Hull-White (extended **Vasicek**(p. 669)) model class.

This class implements the standard single-factor Hull-White model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t$$

where α and σ are constants.

Public Member Functions

- **HullWhite** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.01)
- **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Return by default a trinomial recombining tree.

- **Handle**< ShortRateDynamics > **dynamics** () const

  returns the short-rate dynamics

- double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const

Protected Member Functions

- void **generateArguments** ()
- double **A** (**Time** t, **Time** T) const

The documentation for this class was generated from the following files:

- ql/ShortRateModels/OneFactorModels/**hullwhite.hpp**
- ql/ShortRateModels/OneFactorModels/**hullwhite.cpp**

11.196 HullWhite::Dynamics Class Reference

`#include <hullwhite.hpp>`

11.196.1 Detailed Description

Short-rate dynamics in the Hull-White model.

The short-rate is here

$$r_t = \varphi(t) + x_t$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

Public Member Functions

- **Dynamics** (const **Parameter** &fitting, double a, double sigma)
- double **variable** (**Time** t, **Rate** r) const
- double **shortRate** (**Time** t, double x) const

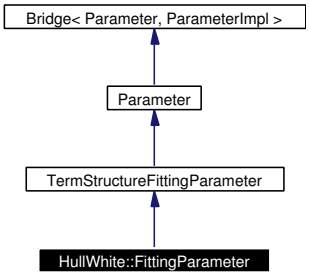
The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**hullwhite.hpp**

11.197 HullWhite::FittingParameter Class Reference

`#include <hullwhite.hpp>`

Inheritance diagram for HullWhite::FittingParameter:



11.197.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2}[\frac{\sigma(1-e^{-at})}{a}]^2,$$

where $f(t)$ is the instantaneous forward rate at t.

Public Member Functions

- **FittingParameter** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a, double sigma)

The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**hullwhite.hpp**

11.198 ICGaussianRng Class Template Reference

`#include <inversecumgaussianrng.hpp>`

11.198.1 Detailed Description

template<class RNG, class I> class QuantLib::RandomNumbers::ICGaussianRng< RNG, I >

Inverse cumulative Gaussian random number generator.

It uses a uniform deviate in (0, 1) as the source of cumulative normal distribution values. Then an inverse cumulative normal distribution is used as it is approximately a Gaussian deviate with average 0.0 and standard deviation 1.0.

The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

The inverse cumulative normal distribution is supplied by I.

Class I must implement the following interface:

```
I::I();
double I::operator() const;
```

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **ICGaussianRng** (const RNG &uniformGenerator)
- **ICGaussianRng** (long seed=0)
- **sample_type next** () const

  returns next sample from the Gaussian distribution

11.198.2 Constructor & Destructor Documentation

11.198.2.1 ICGaussianRng (long *seed* = 0) [explicit]

Deprecated
initialize with a random number generator instead.

The documentation for this class was generated from the following file:

- ql/RandomNumbers/**inversecumgaussianrng.hpp**

11.199 ICGaussianRsg Class Template Reference

`#include <inversecumgaussianrsg.hpp>`

11.199.1 Detailed Description

template<class USG, class I> class QuantLib::RandomNumbers::ICGaussianRsg< USG, I >

Inverse cumulative Gaussian random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative normal distribution values. Then an inverse cumulative normal distribution is used as it is approximately a Gaussian deviate with average 0.0 and standard deviation 1.0.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::next() const;
Size USG::dimension() const;
```

The inverse cumulative normal distribution is supplied by I.

Class I must implement the following interface:

```
I::I();
double I::operator() const;
```

Public Types

- typedef **MonteCarlo::Sample< Array > sample_type**

Public Member Functions

- **ICGaussianRsg** (const USG &uniformSequenceGenerator)
- const **sample_type** & **nextSequence** () const

  returns next sample from the Gaussian distribution

- const **sample_type** & **lastSequence** () const
- **Size dimension** () const

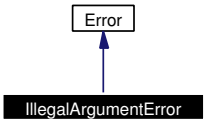
The documentation for this class was generated from the following file:

- ql/RandomNumbers/**inversecumgaussianrsg.hpp**

11.200 IllegalArgumentError Class Reference

`#include <errors.hpp>`

Inheritance diagram for IllegalArgumentError:



11.200.1 Detailed Description

Specialized error.

Thrown upon passing an argument with an illegal value.

Public Member Functions

- **IllegalArgumentError** (const std::string &what="")

The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.201 IllegalResultError Class Reference

`#include <errors.hpp>`

Inheritance diagram for IllegalResultError:

11.201.1 Detailed Description

Specialized error.

Thrown upon obtaining a result outside the allowed range.

Public Member Functions

- **IllegalResultError** (const std::string &what="")

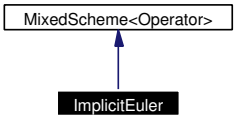
The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.202 ImplicitEuler Class Template Reference

`#include <impliciteuler.hpp>`

Inheritance diagram for ImplicitEuler:



11.202.1 Detailed Description

template<class Operator> class QuantLib::FiniteDifferences::ImplicitEuler< Operator >

Backward Euler scheme for finite difference methods.

See sect. **The finite differences framework**(p. 31) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Friends

- class **FiniteDifferenceModel< ImplicitEuler< Operator > >**

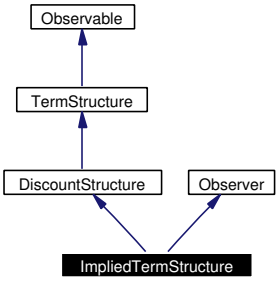
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**impliciteuler.hpp**

11.203 ImpliedTermStructure Class Reference

`#include <impliedtermstructure.hpp>`

Inheritance diagram for ImpliedTermStructure:



11.203.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Public Member Functions

- **ImpliedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **Date** &newTodaysDate, const **Date** &newReferenceDate)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date todaysDate** () const
  today's date
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
  the latest date for which the curve can return rates
- **Time maxTime** () const
  the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const

  returns the discount factor as seen from the evaluation date

11.203.2 Member Function Documentation

11.203.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

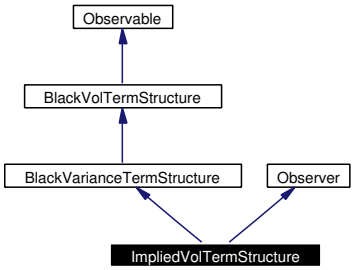
The documentation for this class was generated from the following file:

- ql/TermStructures/**impliedtermstructure.hpp**

11.204 ImpliedVolTermStructure Class Reference

`#include <impliedvoltermstructure.hpp>`

Inheritance diagram for ImpliedVolTermStructure:



11.204.1 Detailed Description

Implied vol term structure at a given date in the future.

The given date will be the implied reference date.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Warning:
It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only

Public Member Functions

- **ImpliedVolTermStructure** (const **RelinkableHandle**< **BlackVolTermStructure** > &originalTS, const **Date** &newReferenceDate)

BlackVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **blackVarianceImpl** (**Time** t, double strike, bool extrapolate=false) const

  implements the actual Black variance calculation in derived classes

11.204.2 Member Function Documentation

11.204.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

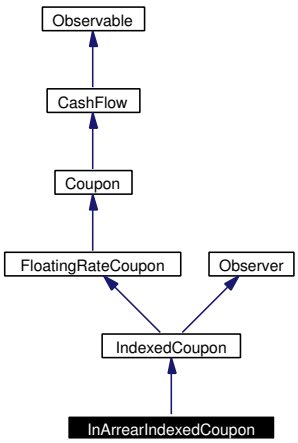
The documentation for this class was generated from the following file:

- ql/Volatilities/**impliedvoltermstructure.hpp**

11.205 InArrearIndexedCoupon Class Reference

`#include <inarrearindexedcoupon.hpp>`

Inheritance diagram for InArrearIndexedCoupon:



11.205.1 Detailed Description

in arrear indexed coupon class

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **InArrearIndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

FloatingRateCoupon interface

- **Date fixingDate** () const

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

The documentation for this class was generated from the following file:

- ql/CashFlows/**inarrearindexedcoupon.hpp**

11.206 IncrementalStatistics Class Reference

`#include <incrementalstatistics.hpp>`

11.206.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

Warning:
high moments are numerically unstable for high average/standardDeviation ratios

Public Member Functions

Inspectors

- **Size samples** () const
  number of samples collected
- double **weightSum** () const
  sum of data weights
- double **mean** () const
- double **variance** () const
- double **standardDeviation** () const
- double **downsideVariance** () const
- double **downsideDeviation** () const
- double **errorEstimate** () const
- double **skewness** () const
- double **kurtosis** () const
- double **min** () const
- double **max** () const

Modifiers

- void **add** (double value, double weight=1.0)
  adds a datum to the set, possibly with a weight
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
  adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
  adds a sequence of data to the set, each with its weight
- void **reset** ()
  resets the data to a null set

Protected Attributes

- **Size sampleNumber_**
- **Size downsideSampleNumber_**
- double **sampleWeight_**
- double **downsideSampleWeight_**
- double **sum_**
- double **quadraticSum_**
- double **downsideQuadraticSum_**
- double **cubicSum_**
- double **fourthPowerSum_**
- double **min_**
- double **max_**

11.206.2 Member Function Documentation

11.206.2.1 double mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

11.206.2.2 double variance () const

returns the variance, defined as

$$\frac{N}{N-1}\left\langle (x - \langle x \rangle)^2 \right\rangle.$$

11.206.2.3 double standardDeviation () const

returns the standard deviation σ, defined as the square root of the variance.

11.206.2.4 double downsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^{N} \theta \times x_i^2}{\sum_{i=1}^{N} w_i}$$

, where θ = 0 if x > 0 and θ =1 if x <0

11.206.2.5 double downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

11.206.2.6 double errorEstimate () const

returns the error estimate ϵ, defined as the square root of the ratio of the variance to the number of samples.

11.206.2.7 double skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\left\langle (x - \langle x \rangle)^3 \right\rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

11.206.2.8 double kurtosis () const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\left\langle (x - \langle x \rangle)^4 \right\rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

11.206.2.9 double min () const

returns the minimum sample value

11.206.2.10 double max () const

returns the maximum sample value

11.206.2.11 void add (double *value*, double *weight* = 1.0)

adds a datum to the set, possibly with a weight

Precondition:
weights must be positive or null

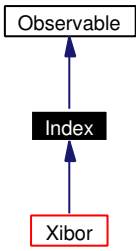
The documentation for this class was generated from the following files:

- ql/Math/**incrementalstatistics.hpp**
- ql/Math/**incrementalstatistics.cpp**

11.207 Index Class Reference

`#include <index.hpp>`

Inheritance diagram for Index:



11.207.1 Detailed Description

purely virtual base class for indexes

Public Member Functions

- virtual std::string **name** () const =0

  Returns the name of the index.

- virtual **Rate fixing** (const **Date** &fixingDate) const =0

  returns the fixing at the given date

11.207.2 Member Function Documentation

11.207.2.1 virtual std::string name () const [pure virtual]

Returns the name of the index.

Warning:
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implemented in **Xibor** (p. 676).

11.207.2.2 virtual Rate fixing (const Date & *fixingDate*) const [pure virtual]

returns the fixing at the given date

Note:
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implemented in **Xibor** (p. 676).

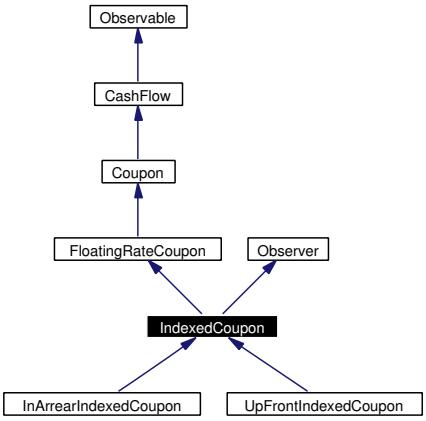
The documentation for this class was generated from the following file:

- ql/**index.hpp**

11.208 IndexedCoupon Class Reference

`#include <indexedcoupon.hpp>`

Inheritance diagram for IndexedCoupon:



11.208.1 Detailed Description

base indexed coupon class

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **IndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

CashFlow interface

- double **amount** () const
  returns the amount of the cash flow

Coupon interface

- **DayCounter dayCounter** () const
  day counter for accrual calculation

FloatingRateCoupon interface

- **Rate fixing** () const

Inspectors

- const **Handle**< **Indexes::Xibor** > & **index** () const

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.208.2 Member Function Documentation

11.208.2.1 double amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. 252).

11.208.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

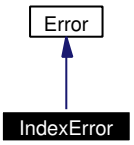
The documentation for this class was generated from the following file:

- ql/CashFlows/**indexedcoupon.hpp**

11.209 IndexError Class Reference

`#include <errors.hpp>`

Inheritance diagram for IndexError:



11.209.1 Detailed Description

Specialized error.

Thrown upon accessing an array or container outside its range.

Public Member Functions

- **IndexError** (const std::string &what="")

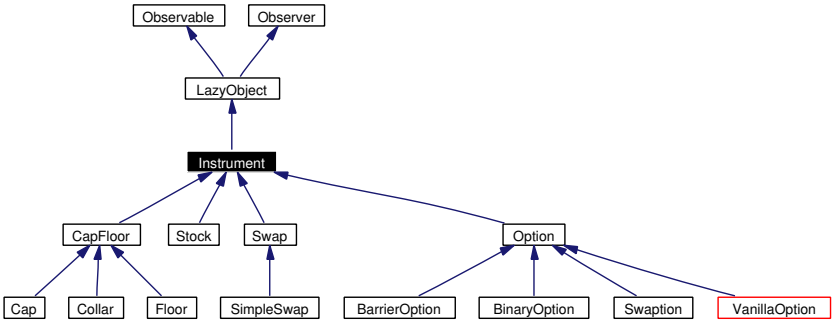
The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.210 Instrument Class Reference

`#include <instrument.hpp>`

Inheritance diagram for Instrument:



11.210.1 Detailed Description

Abstract instrument class.

This class is purely abstract and defines the interface of concrete instruments which will be derived from this one.

Public Member Functions

- **Instrument** (const std::string &isinCode="", const std::string &description="")
- virtual void **setupArguments** (**Arguments** *) const

Inspectors

- std::string **isinCode** () const
  returns the ISIN code of the instrument, when given.
- std::string **description** () const
  returns a brief textual description of the instrument.
- double **NPV** () const
  returns the net present value of the instrument.
- double **errorEstimate** () const
  returns the error estimate on the NPV when available.
- virtual bool **isExpired** () const =0
  returns whether the instrument is still tradable.

Modifiers

- void **setPricingEngine** (const **Handle**< **PricingEngine** > &)
  set the pricing engine to be used.

Protected Member Functions

Calculations

- void **calculate** () const
- virtual void **setupExpired** () const
- virtual void **performCalculations** () const

Protected Attributes

- **Handle< PricingEngine > engine_**

Results

The value of this attribute and any other that derived classes might declare must be set during calculation.

- double **NPV_**
- double **errorEstimate_**

11.210.2 Member Function Documentation

11.210.2.1 void setPricingEngine (const Handle< PricingEngine > &)

set the pricing engine to be used.

Warning:
calling this method will have no effects in case the **performCalculation** method was overridden in a derived class.

11.210.2.2 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented in **BarrierOption** (p. 183), **BinaryOption** (p. 198), **CapFloor** (p. 248), **ForwardVanillaOption** (p. 366), **QuantoForwardVanillaOption** (p. 567), **QuantoVanillaOption** (p. 573), **SimpleSwap** (p. 600), **Swaption** (p. 625), and **VanillaOption** (p. 665).

11.210.2.3 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:
Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.
Should this method be redefined in derived classes, LazyObject::calculate() should be called in the overriding method.

Reimplemented from **LazyObject** (p. 444).

11.210.2.4 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented in **BarrierOption** (p. 183), **BinaryOption** (p. 198), **QuantoVanillaOption** (p. 573), **Swap** (p. 622), and **VanillaOption** (p. 666).

11.210.2.5 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Implements **LazyObject** (p. 444).

Reimplemented in **BarrierOption** (p. 183), **BinaryOption** (p. 198), **ForwardVanillaOption** (p. 366), **QuantoVanillaOption** (p. 573), **Stock** (p. 615), **Swap** (p. 622), **Swaption** (p. 626), and **VanillaOption** (p. 666).

The documentation for this class was generated from the following file:

- ql/**instrument.hpp**

11.211 IntegerFormatter Class Reference

`#include <dataformatters.hpp>`

11.211.1 Detailed Description

Formats integers for output.

Static Public Member Functions

- std::string **toString** (long l, int digits=0)
- std::string **toOrdinal** (long l)
- std::string **toPowerOfTwo** (unsigned long l, int digits=0)

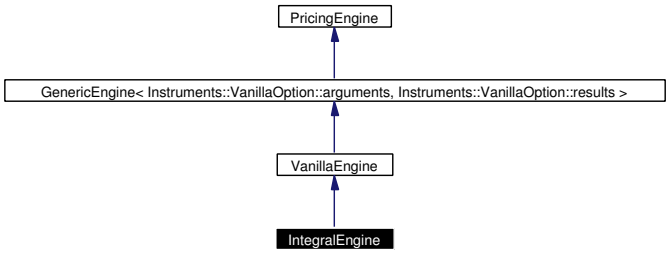
The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.212 IntegralEngine Class Reference

`#include <vanillaengines.hpp>`

Inheritance diagram for IntegralEngine:



11.212.1 Detailed Description

Pricing engine for Vanilla options using integral approach.

Public Member Functions

- void **calculate** () const

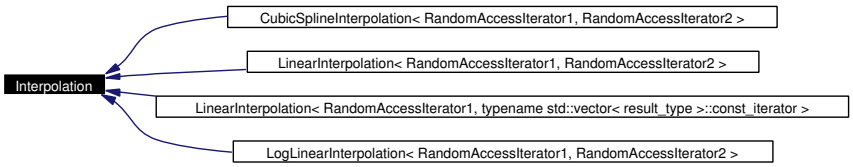
The documentation for this class was generated from the following files:

- ql/PricingEngines/**vanillaengines.hpp**
- ql/PricingEngines/**integralengines.cpp**

11.213 Interpolation Class Template Reference

`#include <interpolation.hpp>`

Inheritance diagram for Interpolation:



11.213.1 Detailed Description

template<class RandomAccessIterator1, class RandomAccessIterator2> class QuantLib::Math::Interpolation< RandomAccessIterator1, RandomAccessIterator2 >

abstract base class for 1-D interpolations

Classes derived from this class will override operator() to provide interpolated values from two sequences of equal length, representing discretized values of a variable and a function of the former, respectively.

Public Member Functions

- **Interpolation** (const RandomAccessIterator1 &xBegin, const RandomAccessIterator1 &xEnd, const RandomAccessIterator2 &yBegin)
- virtual result_type **operator()** (const argument_type &x, bool allowExtrapolation=false) const =0

Public Attributes

- typedef< RandomAccessIterator1 >::value_type **argument_type**
- typedef< RandomAccessIterator2 >::value_type **result_type**

Protected Member Functions

- void **locate** (const argument_type &x) const

Protected Attributes

- bool **isOutOfRange_**
- RandomAccessIterator1 **position_**
- RandomAccessIterator1 **xBegin_**
- RandomAccessIterator1 **xEnd_**
- RandomAccessIterator2 **yBegin_**
- **Size n_**

11.213.2 Member Function Documentation

11.213.2.1 virtual result_type operator() (const argument_type & *x*, bool *allowExtrapolation* = false) const [pure virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implemented in **CubicSplineInterpolation** (p. 284), **LinearInterpolation** (p. 452), **LogLinearInterpolation** (p. 466), and **LinearInterpolation< RandomAccessIterator1, typename std::vector< result_type >::const_iterator >** (p. 452).

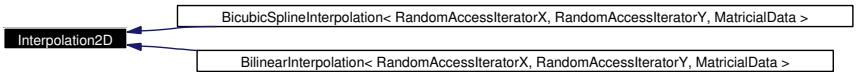
The documentation for this class was generated from the following file:

- ql/Math/**interpolation.hpp**

11.214 Interpolation2D Class Template Reference

`#include <interpolation2D.hpp>`

Inheritance diagram for Interpolation2D:



11.214.1 Detailed Description

template<class RandomAccessIteratorX, class RandomAccessIteratorY, class MatricialData> class QuantLib::Math::Interpolation2D< RandomAccessIteratorX, RandomAccessIteratorY, MatricialData >

abstract base class for 2-D interpolations

Classes derived from this class will override operator() to provide interpolated values from two sequences of length N and M, representing the discretized values of the x,y variables, and a NxM matrix representing the function tabulated z values.

Todo

Bicubic interpolation and bicubic spline

Public Types

- typedef double **result_type**

Public Member Functions

- **Interpolation2D** (const RandomAccessIteratorX &xBegin, const RandomAccessIteratorX &xEnd, const RandomAccessIteratorY &yBegin, const RandomAccessIteratorY &yEnd, const MatricialData &data)
- virtual double **operator()** (const first_argument_type &x, const second_argument_type &y, bool allowExtrapolation=false) const =0

Public Attributes

- typedef< RandomAccessIteratorX >::value_type **first_argument_type**
- typedef< RandomAccessIteratorY >::value_type **second_argument_type**

Protected Member Functions

- void **locate** (const first_argument_type &x, const second_argument_type &y) const

Protected Attributes

- bool **isOutOfRange_**
- RandomAccessIteratorX **xPos_**
- RandomAccessIteratorY **yPos_**
- RandomAccessIteratorX **xBegin_**
- RandomAccessIteratorX **xEnd_**
- RandomAccessIteratorY **yBegin_**
- RandomAccessIteratorY **yEnd_**
- const MatricialData & **data_**

11.214.2 Member Function Documentation

11.214.2.1 virtual double operator() (const first_argument_type & *x*, const second_argument_type & *y*, bool *allowExtrapolation* = false) const [pure virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implemented in **BicubicSplineInterpolation** (p. 191), and **BilinearInterpolation** (p. 193).

The documentation for this class was generated from the following file:

- ql/Math/**interpolation2D.hpp**

11.215 InverseCumulativeNormal Class Reference

`#include <normaldistribution.hpp>`

11.215.1 Detailed Description

Inverse cumulative normal distribution function.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It use Acklam's approximation: by Peter J. Acklam, University of Oslo, Statistics Division. URL: `http://home.online.no/~pjacklam/notes/invnorm/index.html`

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Public Member Functions

- **InverseCumulativeNormal** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const

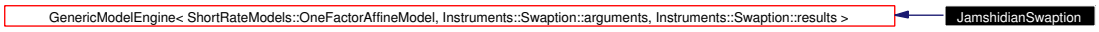
The documentation for this class was generated from the following files:

- ql/Math/**normaldistribution.hpp**
- ql/Math/**normaldistribution.cpp**

11.216 JamshidianSwaption Class Reference

`#include <jamshidianswaption.hpp>`

Inheritance diagram for JamshidianSwaption:



11.216.1 Detailed Description

Jamshidian swaption pricer.

Public Member Functions

- **JamshidianSwaption** (const **Handle**< **ShortRateModels::OneFactorAffineModel** > &modl)
- void **calculate** () const

Friends

- class **rStarFinder**

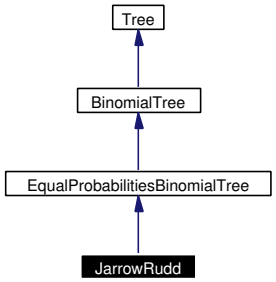
The documentation for this class was generated from the following files:

- ql/Pricers/**jamshidianswaption.hpp**
- ql/Pricers/**jamshidianswaption.cpp**

11.217 JarrowRudd Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for JarrowRudd:



11.217.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

Public Member Functions

- **JarrowRudd** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)

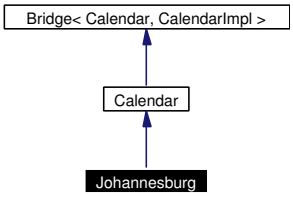
The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.218 Johannesburg Class Reference

`#include <johannesburg.hpp>`

Inheritance diagram for Johannesburg:



11.218.1 Detailed Description

Johannesburg calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Family Day, Easter Monday
- Human Rights Day, March 21st (possibly moved to Monday)
- Freedom Day, April 27th (possibly moved to Monday)
- Workers Day, May 1st (possibly moved to Monday)
- Youth Day, June 16th (possibly moved to Monday)
- National Women's Day, August 9th (possibly moved to Monday)
- Heritage Day, September 24th (possibly moved to Monday)
- Day of Reconciliation, December 16th (possibly moved to Monday)
- Christmas December 25th
- Day of Goodwill December 26th (possibly moved to Monday)

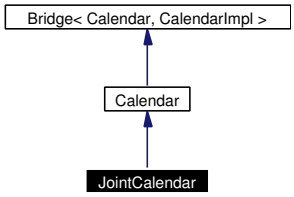
The documentation for this class was generated from the following file:

- ql/Calendars/**johannesburg.hpp**

11.219 JointCalendar Class Reference

`#include <jointcalendar.hpp>`

Inheritance diagram for JointCalendar:



11.219.1 Detailed Description

Joint calendar.

Depending on the chosen rule, this calendar has a set of business days given by either the union or the intersection of the sets of business days of the given calendars.

Public Member Functions

- **JointCalendar** (const **Calendar** &, const **Calendar** &, **JointCalendarRule**=JoinHolidays)
- **JointCalendar** (const **Calendar** &, const **Calendar** &, const **Calendar** &, **JointCalendarRule**=JoinHolidays)
- **JointCalendar** (const **Calendar** &, const **Calendar** &, const **Calendar** &, const **Calendar** &, **JointCalendarRule**=JoinHolidays)

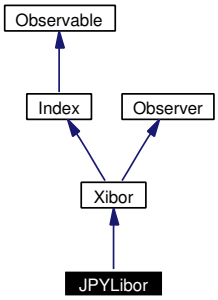
The documentation for this class was generated from the following files:

- ql/Calendars/**jointcalendar.hpp**
- ql/Calendars/**jointcalendar.cpp**

11.220 JPYLibor Class Reference

`#include <jpylibor.hpp>`

Inheritance diagram for JPYLibor:



11.220.1 Detailed Description

JPY Libor index (Also known as TIBOR, check settlement days)

Public Member Functions

- **JPYLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual360**())

The documentation for this class was generated from the following file:

- ql/Indexes/**jpylibor.hpp**

11.221 KnuthUniformRng Class Reference

`#include <knuthuniformrng.hpp>`

11.221.1 Detailed Description

Uniform random number generator.

Random number generator by Knuth. For more details see Knuth, Seminumerical Algorithms, 3rd edition, Section 3.6.

Note:

This is **not** Knuth's original implementation which is available at `http://www-cs-faculty.stanford.edu/~knuth/programs.html`, but rather a slightly modified version wrapped in a C++ class. Such modifications did not affect the code but only the data structures used, which were converted to their standard C++ equivalents.

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **KnuthUniformRng** (long seed=0)
- **sample_type next** () const

11.221.2 Constructor & Destructor Documentation

11.221.2.1 KnuthUniformRng (long *seed* = 0) `[explicit]`

if the given seed is 0, a random seed will be chosen based on clock()

11.221.3 Member Function Documentation

11.221.3.1 KnuthUniformRng::sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

The documentation for this class was generated from the following files:

- ql/RandomNumbers/**knuthuniformrng.hpp**
- ql/RandomNumbers/**knuthuniformrng.cpp**

11.222 KronrodIntegral Class Reference

`#include <kronrodintegral.hpp>`

11.222.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

References: Gauss-Kronrod Integration `http://mathcssun1.emporia.edu/~oneilcat/Experiment-Applet3/ExperimentApplet3.html` NMS - Numerical Analysis Library `http://www.math.iastate.edu/burkardt/f_src/nms/nms.html`

Public Member Functions

- **KronrodIntegral** (double tolerance, **Size** maxFunctionEvaluations=**Null**< int >())
- template<class F> double **operator()** (const F &f, double a, double b) const
- **Size functionEvaluations** ()

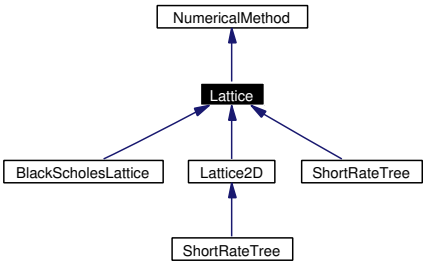
The documentation for this class was generated from the following file:

- ql/Math/**kronrodintegral.hpp**

11.223 Lattice Class Reference

`#include <lattice.hpp>`

Inheritance diagram for Lattice:



11.223.1 Detailed Description

Lattice-method base class.

This class defines a lattice method that is able to rollback (with discount) a discretized asset object. It will usually be based on one or more trees.

Public Member Functions

- **Lattice** (const **TimeGrid** &timeGrid, **Size** n)
- double **presentValue** (const **Handle**< **DiscretizedAsset** > &asset)

  Computes the present value of an asset using Arrow-Debrew prices.

- void **initialize** (const **Handle**< **DiscretizedAsset** > &asset, **Time** t) const

  Initialize a **DiscretizedAsset**(p. 308) *object.*

- void **rollback** (const **Handle**< **DiscretizedAsset** > &asset, **Time** to) const
- void **rollAlmostBack** (const **Handle**< **DiscretizedAsset** > &asset, **Time** to) const
- virtual **Size size** (**Size** i) const =0
- virtual **DiscountFactor discount** (**Size** i, **Size** index) const =0

  Discount factor at time t_i and node indexed by index.

- const **Array** & **statePrices** (**Size** i)
- virtual **Size descendant** (**Size** i, **Size** index, **Size** branch) const =0

  Tree(p. 648) *properties.*

- virtual double **probability** (**Size** i, **Size** index, **Size** branch) const =0

Protected Member Functions

- void **computeStatePrices** (**Size** until)
- virtual void **stepback** (**Size** i, const **Array** &values, **Array** &newValues) const

Protected Attributes

- std::vector< **Array** > **statePrices_**

11.223.2 Member Function Documentation

11.223.2.1 void rollback (const Handle< DiscretizedAsset > & *asset*, Time *to*) const [virtual]

Roll back a **DiscretizedAsset**(p. 308) object until a certain time, performing any needed adjustment

Implements **NumericalMethod** (p. 522).

11.223.2.2 void rollAlmostBack (const Handle< DiscretizedAsset > & *asset*, Time *to*) const [virtual]

Roll-back a **DiscretizedAsset**(p. 308) object until a certain time, but do not perform the final adjustment.

Implements **NumericalMethod** (p. 522).

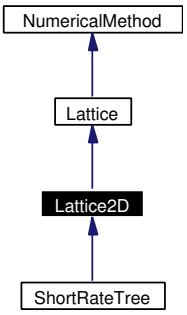
The documentation for this class was generated from the following files:

- ql/Lattices/**lattice.hpp**
- ql/Lattices/**lattice.cpp**

11.224 Lattice2D Class Reference

`#include <lattice2d.hpp>`

Inheritance diagram for Lattice2D:



11.224.1 Detailed Description

Two-dimensional lattice.

This lattice is based on two trinomial trees and primarly used for the G2 short-rate model.

Public Member Functions

- **Lattice2D** (const **Handle**< **TrinomialTree** > &tree1, const **Handle**< **TrinomialTree** > &tree2, double correlation)
- **Size size** (**Size** i) const

Protected Member Functions

- **Size descendant** (**Size** i, **Size** index, **Size** branch) const

  Tree(p. 648) *properties.*

- double **probability** (**Size** i, **Size** index, **Size** branch) const

Protected Attributes

- **Handle**< **Tree** > **tree1_**
- **Handle**< **Tree** > **tree2_**

The documentation for this class was generated from the following files:

- ql/Lattices/**lattice2d.hpp**
- ql/Lattices/**lattice2d.cpp**

11.225 LatticeShortRateModelEngine Class Template Reference

`#include <latticeshortratemodelengine.hpp>`

Inheritance diagram for LatticeShortRateModelEngine:

11.225.1 Detailed Description

template<class Arguments, class Results> class QuantLib::PricingEngines::LatticeShortRateModelEngine< Arguments, Results >

Engine for a short-rate model specialized on a lattice.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **LatticeShortRateModelEngine** (const **Handle**< **ShortRateModels::Model** > &model, **Size** timeSteps)
- **LatticeShortRateModelEngine** (const **Handle**< **ShortRateModels::Model** > &model, const **TimeGrid** &timeGrid)
- void **update** ()

Protected Attributes

- **TimeGrid timeGrid_**
- **Size timeSteps_**
- **Handle**< **Lattices::Lattice** > **lattice_**

11.225.2 Member Function Documentation

11.225.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from **GenericModelEngine** (p. 382).

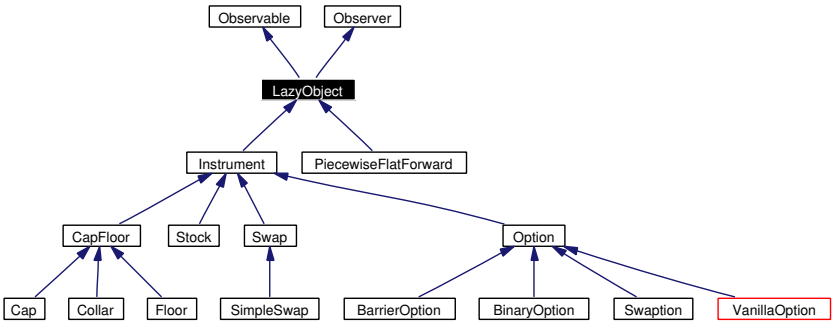
The documentation for this class was generated from the following file:

- ql/PricingEngines/**latticeshortratemodelengine.hpp**

11.226 LazyObject Class Reference

`#include <lazyobject.hpp>`

Inheritance diagram for LazyObject:



11.226.1 Detailed Description

Framework for calculation on demand and result caching.

Calculations

These methods do not modify the structure of the object and are therefore declared as `const`. Data members which will be calculated on demand need to be declared as mutable.

- void **recalculate** ()
- void **freeze** ()
- void **unfreeze** ()
- virtual void **calculate** () const
- virtual void **performCalculations** () const =0

Public Member Functions

Observer interface

- void **update** ()

Protected Attributes

- bool **calculated_**
- bool **frozen_**

11.226.2 Member Function Documentation

11.226.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.226.2.2 void recalculate ()

This method force the recalculation of any results which would otherwise be cached. It is not declared as `const` since it needs to call the non-`const` **notifyObservers** method.

Note:

Explicit invocation of this method is **not** necessary if the object registered itself as observer with the structures on which such results depend. It is strongly advised to follow this policy when possible.

11.226.2.3 void freeze ()

This method constrains the object to return the presently cached results on successive invocations, even if arguments upon which they depend should change.

11.226.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

11.226.2.5 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.
Should this method be redefined in derived classes, **LazyObject::calculate()**(p. 444) should be called in the overriding method.

Reimplemented in **Instrument** (p. 423).

11.226.2.6 virtual void performCalculations () const [protected, pure virtual]

This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in **Instrument** (p. 424), **BarrierOption** (p. 183), **BinaryOption** (p. 198), **ForwardVanillaOption** (p. 366), **QuantoVanillaOption** (p. 573), **Stock** (p. 615), **Swap** (p. 622), **Swaption** (p. 626), and **VanillaOption** (p. 666).

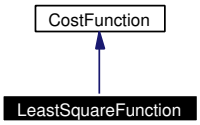
The documentation for this class was generated from the following file:

- ql/Patterns/**lazyobject.hpp**

11.227 LeastSquareFunction Class Reference

`#include <leastsquare.hpp>`

Inheritance diagram for LeastSquareFunction:



11.227.1 Detailed Description

Design a least square function as a cost function using the interface provided by **LeastSquareProblem**(p. 447) class. **Array**(p. 173) vector class requires function DotProduct() that computes dot product and - operator. M matrix class requires function transpose() that computes transpose and * operator with vector class.

Public Member Functions

- **LeastSquareFunction** (**LeastSquareProblem** &lsp)
  Default constructor.
- virtual ~**LeastSquareFunction** ()
  Destructor.
- virtual double **value** (const **Array** &x) const
  compute value of the least square function
- virtual void **gradient** (**Array** &grad_f, const **Array** &x) const
  compute vector of derivatives of the least square function
- virtual double **valueAndGradient** (**Array** &grad_f, const **Array** &x) const
  compute value and gradient of the least square function

Protected Attributes

- **LeastSquareProblem** & **lsp_**
  least square problem

The documentation for this class was generated from the following file:

- ql/Optimization/**leastsquare.hpp**

11.228 LeastSquareProblem Class Reference

`#include <leastsquare.hpp>`

11.228.1 Detailed Description

Base class for least square problem.

Public Member Functions

- virtual int **size** ()=0
  size of the problem ie size of target vector
- virtual void **targetAndValue** (const **Array** &x, **Array** &target, **Array** &fct2fit)=0
  compute the target vector and the values of the fonction to fit
- virtual void **targetValueAndGradient** (const **Array** &x, **Math::Matrix** &grad_fct2fit, **Array** &target, **Array** &fct2fit)=0

11.228.2 Member Function Documentation

11.228.2.1 virtual void targetValueAndGradient (const Array & *x*, Math::Matrix & *grad_fct2fit*, Array & *target*, Array & *fct2fit*) [pure virtual]

compute the target vector, the values of the fonction to fit and the matrix of derivatives

The documentation for this class was generated from the following file:

- ql/Optimization/**leastsquare.hpp**

11.229 LecuyerUniformRng Class Reference

`#include <lecuyeruniformrng.hpp>`

11.229.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (know as ran2 in Numerical recipes)

For more details see Section 7.1 of Numerical Recipes in C, 2nd Edition, Cambridge University Press (available at `http://www.nr.com/`)

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **LecuyerUniformRng** (long seed=0)
- **sample_type next** () const

11.229.2 Constructor & Destructor Documentation

11.229.2.1 LecuyerUniformRng (long *seed* = 0) `[explicit]`

if the given seed is 0, a random seed will be chosen based on clock()

11.229.3 Member Function Documentation

11.229.3.1 LecuyerUniformRng::sample_type next () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

The documentation for this class was generated from the following files:

- ql/RandomNumbers/**lecuyeruniformrng.hpp**
- ql/RandomNumbers/**lecuyeruniformrng.cpp**

11.230 LexicographicalView Class Template Reference

`#include <lexicographicalview.hpp>`

11.230.1 Detailed Description

template<class RandomAccessIterator> class QuantLib::Math::LexicographicalView< RandomAccessIterator >

Lexicographical 2-D view of a contiguous set of data.

This view can be used to easily store a discretized 2-D function in an array to be used in a finite differences calculation.

Public Types

- typedef RandomAccessIterator **x_iterator**
  iterates over v_{ij} with j fixed.
- typedef **Utilities::stepping_iterator**< RandomAccessIterator > **y_iterator**
  iterates over v_{ij} with i fixed.

Public Member Functions

- **LexicographicalView** (const RandomAccessIterator &begin, const RandomAccessIterator &end, int xSize)
  attaches the view with the given dimension to a sequence
- typedef **QL_REVERSE_ITERATOR** (RandomAccessIterator, typename 1< RandomAccess-Iterator >::value_type) reverse_x_iterator
  iterates backwards over v_{ij} with j fixed.
- typedef **QL_REVERSE_ITERATOR** (**y_iterator**, typename 1< RandomAccessIterator >::value_type) reverse_y_iterator
  iterates backwards over v_{ij} with i fixed.

Element access

- **y_iterator operator[ ]** (int i)

Iterator access

- **x_iterator xbegin** (int j)
- **x_iterator xend** (int j)
- reverse_x_iterator **rxbegin** (int j)
- reverse_x_iterator **rxend** (int j)
- **y_iterator ybegin** (int i)
- **y_iterator yend** (int i)
- reverse_y_iterator **rybegin** (int i)

- reverse_y_iterator **ryend** (int i)

Inspectors

- int **xSize** () const
  dimension of the array along x
- int **ySize** () const
  dimension of the array along y

The documentation for this class was generated from the following file:

- ql/Math/**lexicographicalview.hpp**

11.231 Linear Class Reference

`#include <interpolationtraits.hpp>`

11.231.1 Detailed Description

linear interpolation traits

Static Public Member Functions

- template<class I1, class I2> **Handle**< **Interpolation**< I1, I2 > > **make_interpolation** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)
- template<class I1, class I2, class M> **Handle**< **Interpolation2D**< I1, I2, M > > **make_interpolation** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z)

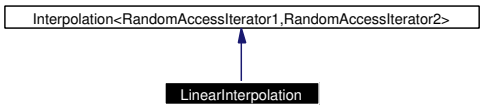
The documentation for this class was generated from the following file:

- ql/Math/**interpolationtraits.hpp**

11.232 LinearInterpolation Class Template Reference

`#include <linearinterpolation.hpp>`

Inheritance diagram for LinearInterpolation:



11.232.1 Detailed Description

template<class RandomAccessIterator1, class RandomAccessIterator2> class QuantLib::Math::LinearInterpolation< RandomAccessIterator1, RandomAccessIterator2 >

linear interpolation between discrete points

Public Member Functions

- **LinearInterpolation** (const RandomAccessIterator1 &xBegin, const RandomAccessIterator1 &xEnd, const RandomAccessIterator2 &yBegin)
- result_type **operator()** (const argument_type &x, bool allowExtrapolation=false) const

Public Attributes

- typedef< RandomAccessIterator1 >::value_type **argument_type**
- typedef< RandomAccessIterator2 >::value_type **result_type**

11.232.2 Member Function Documentation

11.232.2.1 result_type operator() (const argument_type & *x*, bool *allowExtrapolation* = false) const [virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implements **Interpolation** (p. 428).

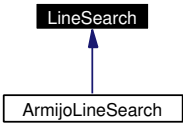
The documentation for this class was generated from the following file:

- ql/Math/**linearinterpolation.hpp**

11.233 LineSearch Class Reference

`#include <linesearch.hpp>`

Inheritance diagram for LineSearch:



11.233.1 Detailed Description

Base class for line search.

Public Member Functions

- **LineSearch** (double eps=1e-8)
  Default constructor.
- virtual ~**LineSearch** ()
  Destructor.
- const **Array** & **lastX** ()
  return last x value
- double **lastFunctionValue** ()
  return last cost function value
- const **Array** & **lastGradient** ()
  return last gradient
- double **lastGradientNorm2** ()
  return square norm of last gradient
- bool **succeed** ()
- virtual double **operator()** (const **Problem** &P, double t_ini)=0
  Perform line search.
- double **update** (**Array** &params, const **Array** &direction, double beta, const **Constraint** &constraint)

Protected Attributes

- **Array xtd_**
  new x and its gradient

- **Array gradient_**
  new x and its gradient

- double **qt_**
  *cost function value and gradient norm corresponding to xtd_*

- double **qpt_**
  *cost function value and gradient norm corresponding to xtd_*

- bool **succeed_**
  flag to know if linesearch succeed

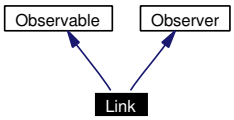
The documentation for this class was generated from the following file:

- ql/Optimization/**linesearch.hpp**

11.234 Link Class Template Reference

`#include <relinkablehandle.hpp>`

Inheritance diagram for Link:



11.234.1 Detailed Description

template<class Type> class QuantLib::Link< Type >

Relinkable access to a **Handle**(p. 390).

Precondition:
Class "Type" must inherit from Observable

Public Member Functions

- **Link** (const **Handle**< Type > &h=**Handle**< Type >(), bool registerAsObserver=true)
- void **linkTo** (const **Handle**< Type > &h, bool registerAsObserver=true)
- bool **isNull** () const
  Checks if the contained handle points to anything.
- const **Handle**< Type > & **currentLink** () const
  Returns the contained handle.
- void **update** ()
  Observer interface.

11.234.2 Constructor & Destructor Documentation

11.234.2.1 Link (const Handle< Type > & *h* = Handle< Type >(), bool *registerAsObserver* = true) `[explicit]`

Warning:
see the documentation of the `linkTo` method for issues relatives to `registerAsObserver`.

11.234.3 Member Function Documentation

11.234.3.1 void linkTo (const Handle< Type > & *h*, bool *registerAsObserver* = true)

Warning:
registerAsObserver is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be

set to `false` when the passed handle was created with `owns = false` (the latter should only happen in a controlled environment, so that the programmer is aware of it). Failure to do so can very likely result in a program crash. If the programmer does want the relinkable handle to register as observer of such a handle, it is his responsibility to ensure that the relinkable handle gets destroyed before the pointed object does.

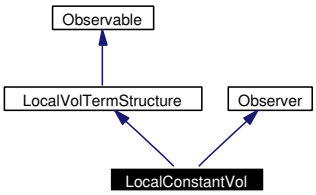
The documentation for this class was generated from the following file:

- ql/**relinkablehandle.hpp**

11.235 LocalConstantVol Class Reference

`#include <localconstantvol.hpp>`

Inheritance diagram for LocalConstantVol:



11.235.1 Detailed Description

Constant local volatility, no time-strike dependence.

This class implements the LocalVolatilityTermStructure interface for a constant local volatility (no time/asset dependence). Local volatility and Black volatility are the same when volatility is at most time dependent, so this class is basically a proxy for **BlackVolatilityTermStructure**(p. 221).

Public Member Functions

- **LocalConstantVol** (const **Date** &referenceDate, double volatility, const **DayCounter** &dayCounter=**DayCounters::Actual365**())
- **LocalConstantVol** (const **Date** &referenceDate, const **RelinkableHandle**< **MarketElement** > &volatility, const **DayCounter** &dayCounter=**DayCounters::Actual365**())

LocalVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.235.2 Member Function Documentation

11.235.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

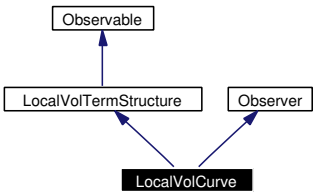
The documentation for this class was generated from the following file:

- ql/Volatilities/**localconstantvol.hpp**

11.236 LocalVolCurve Class Reference

`#include <localvolcurve.hpp>`

Inheritance diagram for LocalVolCurve:



11.236.1 Detailed Description

Local volatility curve derived from a Black curve.

Public Member Functions

- **LocalVolCurve** (const **RelinkableHandle**< **BlackVarianceCurve** > &curve)

LocalVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- double **localVolImpl** (**Time**, double, bool extrapolate) const

11.236.2 Member Function Documentation

11.236.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.236.2.2 double localVolImpl (Time *t*, double *dummy*, bool *extrapolate*) const [protected, virtual]

The relation

$$\int_0^T \sigma_L^2(t)dt = \sigma_B^2 T$$

holds, where $\sigma_L(t)$ is the local volatility at time t and $\sigma_B(T)$ is the Black volatility for maturity T. From the above, the formula

$$\sigma_L(t) = \sqrt{\frac{\mathrm{d}}{\mathrm{d}t}\sigma_B^2(t)t}$$

can be deduced which is here implemented.

Implements **LocalVolTermStructure** (p. 464).

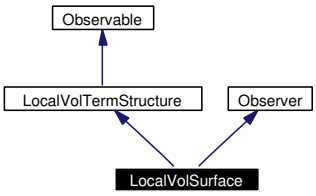
The documentation for this class was generated from the following file:

- ql/Volatilities/**localvolcurve.hpp**

11.237 LocalVolSurface Class Reference

`#include <localvolsurface.hpp>`

Inheritance diagram for LocalVolSurface:



11.237.1 Detailed Description

Local volatility surface derived from a Black vol surface.

For details about this implementation refers to "Stochastic Volatility and Local Volatility", by Jim Gatheral www.math.nyu.edu/fellows_fin_math/gatheral/Lecture1_Fall02.pdf

Public Member Functions

- **LocalVolSurface** (const **RelinkableHandle**< **BlackVolTermStructure** > &blackTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **MarketElement** > &underlying)
- **LocalVolSurface** (const **RelinkableHandle**< **BlackVolTermStructure** > &blackTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &dividendTS, double underlying)

LocalVolTermStructure interface

- **Date referenceDate** () const
  returns the reference date for which t=0
- **DayCounter dayCounter** () const
  returns the day counter
- **Date maxDate** () const
  the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- double **localVolImpl** (**Time**, double, bool extrapolate) const

  implements the actual local vol calculation in derived classes

11.237.2 Member Function Documentation

11.237.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

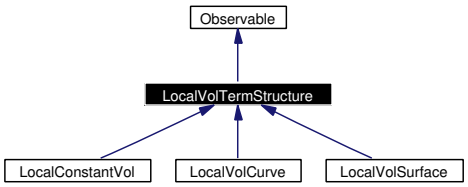
The documentation for this class was generated from the following files:

- ql/Volatilities/**localvolsurface.hpp**
- ql/Volatilities/**localvolsurface.cpp**

11.238 LocalVolTermStructure Class Reference

`#include <voltermstructure.hpp>`

Inheritance diagram for LocalVolTermStructure:



11.238.1 Detailed Description

Local Volatility Term structure.

This abstract class defines the interface of concrete local volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Local Volatility

- double **localVol** (const **Date** &d, double underlyingLevel, bool extrapolate=false) const
- double **localVol** (**Time** t, double underlyingLevel, bool extrapolate=false) const

Dates

- virtual **Date referenceDate** () const =0

  returns the reference date for which t=0

- virtual **DayCounter dayCounter** () const =0

  returns the day counter

- virtual **Date maxDate** () const =0

  the latest date for which the term structure can return vols

- **Time maxTime** () const

  the latest time for which the term structure can return vols

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

Protected Member Functions

- virtual double **localVolImpl** (**Time** t, double strike, bool extrapolate=false) const =0
  implements the actual local vol calculation in derived classes

The documentation for this class was generated from the following files:

- ql/**voltermstructure.hpp**
- ql/**voltermstructure.cpp**

11.239 LogLinear Class Reference

`#include <interpolationtraits.hpp>`

11.239.1 Detailed Description

log-linear interpolation traits

Static Public Member Functions

- template<class I1, class I2> **Handle**< **Interpolation**< I1, I2 > > **make_interpolation** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

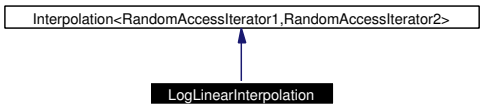
The documentation for this class was generated from the following file:

- ql/Math/**interpolationtraits.hpp**

11.240 LogLinearInterpolation Class Template Reference

`#include <loglinearinterpolation.hpp>`

Inheritance diagram for LogLinearInterpolation:



11.240.1 Detailed Description

template<class RandomAccessIterator1, class RandomAccessIterator2> class QuantLib::Math::LogLinearInterpolation< RandomAccessIterator1, RandomAccessIterator2 >

log linear interpolation between discrete points

Public Member Functions

- **LogLinearInterpolation** (const RandomAccessIterator1 &xBegin, const RandomAccessIterator1 &xEnd, const RandomAccessIterator2 &yBegin)
- result_type **operator()** (const argument_type &x, bool allowExtrapolation=false) const

Public Attributes

- typedef< RandomAccessIterator1 >::value_type **argument_type**
- typedef< RandomAccessIterator2 >::value_type **result_type**

11.240.2 Member Function Documentation

11.240.2.1 result_type operator() (const argument_type & *x*, bool *allowExtrapolation* = false) const [virtual]

This operator must be overridden to provide an implementation of the actual interpolation.

Precondition:
The sequence of values for x must have been sorted for the result to make sense.

Implements **Interpolation** (p. 428).

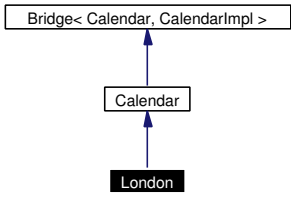
The documentation for this class was generated from the following file:

- ql/Math/**loglinearinterpolation.hpp**

11.241 London Class Reference

`#include <london.hpp>`

Inheritance diagram for London:



11.241.1 Detailed Description

London calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

The documentation for this class was generated from the following file:

- ql/Calendars/**london.hpp**

11.242 lowest_category_iterator Struct Template Reference

`#include <iteratorcategories.hpp>`

11.242.1 Detailed Description

template<class Category1, class Category2> struct QuantLib::Utilities::lowest_category_iterator< Category1, Category2 >

most generic of two given iterator categories

Specializations of this struct define a typedef iterator_category which corresponds to the most generic of the two input categories, e.g., **lowest_category_iterator**(p. 468)<std::random_access_iterator_tag, std::forward_iterator_tag>::iterator_category corresponds to std::forward_iterator_tag.

The documentation for this struct was generated from the following file:

- ql/Utilities/**iteratorcategories.hpp**

11.243 MakeSchedule Class Reference

`#include <scheduler.hpp>`

11.243.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

Public Member Functions

- **MakeSchedule** (const **Calendar** &calendar, const **Date** &startDate, const **Date** &endDate, int frequency, **RollingConvention** rollingConvention, bool isAdjusted)
- **MakeSchedule** & **withStubDate** (const **Date** &d)
- **MakeSchedule** & **backwards** (bool flag=true)
- **MakeSchedule** & **forwards** (bool flag=true)
- **MakeSchedule** & **longFinalPeriod** (bool flag=true)
- **MakeSchedule** & **shortFinalPeriod** (bool flag=true)
- **operator Schedule** ()

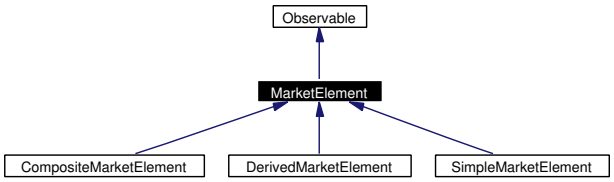
The documentation for this class was generated from the following file:

- ql/**scheduler.hpp**

11.244 MarketElement Class Reference

`#include <marketelement.hpp>`

Inheritance diagram for MarketElement:



11.244.1 Detailed Description

purely virtual base class for market observables

Public Member Functions

- virtual double **value** () const =0

  returns the current value

The documentation for this class was generated from the following file:

- ql/**marketelement.hpp**

11.245 Matrix Class Reference

`#include <matrix.hpp>`

11.245.1 Detailed Description

matrix used in linear algebra.

This class implements the concept of **Matrix**(p. 471) as used in linear algebra. As such, it is **not** meant to be used as a container.

Public Types

- typedef double * **iterator**
- typedef const double * **const_iterator**
- typedef double * **row_iterator**
- typedef const double * **const_row_iterator**
- typedef **Utilities::stepping_iterator**< double * > **column_iterator**
- typedef **Utilities::stepping_iterator**< const double * > **const_column_iterator**

Public Member Functions

- typedef **QL_REVERSE_ITERATOR** (iterator, double) reverse_iterator
- typedef **QL_REVERSE_ITERATOR** (const_iterator, double) const _reverse_iterator
- typedef **QL_REVERSE_ITERATOR** (row_iterator, double) reverse_row_iterator
- typedef **QL_REVERSE_ITERATOR** (const_row_iterator, double) const _reverse_row_-iterator
- typedef **QL_REVERSE_ITERATOR** (**column_iterator**, double) reverse_column_iterator
- typedef **QL_REVERSE_ITERATOR** (**const_column_iterator**, double) const _reverse_-column_iterator

Constructors, destructor, and assignment

- **Matrix** ()
  creates a null matrix
- **Matrix** (**Size** rows, **Size** columns)
  creates a matrix with the given dimensions
- **Matrix** (**Size** rows, **Size** columns, double value)
  creates the matrix and fills it with `value`
- **Matrix** (const **Matrix** &)
- **Matrix** (const **Disposable**< **Matrix** > &)
- **Matrix** & **operator=** (const **Matrix** &)
- **Matrix** & **operator=** (const **Disposable**< **Matrix** > &)

Algebraic operators

- **Matrix** & **operator+=** (const **Matrix** &)
- **Matrix** & **operator-=** (const **Matrix** &)

- **Matrix** & **operator *=** (double)
- **Matrix** & **operator/=** (double)

Iterator access

- const_iterator **begin** () const
- iterator **begin** ()
- const_iterator **end** () const
- iterator **end** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rend** () const
- reverse_iterator **rend** ()
- const_row_iterator **row_begin** (**Size** i) const
- row_iterator **row_begin** (**Size** i)
- const_row_iterator **row_end** (**Size** i) const
- row_iterator **row_end** (**Size** i)
- const_reverse_row_iterator **row_rbegin** (**Size** i) const
- reverse_row_iterator **row_rbegin** (**Size** i)
- const_reverse_row_iterator **row_rend** (**Size** i) const
- reverse_row_iterator **row_rend** (**Size** i)
- **const_column_iterator column_begin** (**Size** i) const
- **column_iterator column_begin** (**Size** i)
- **const_column_iterator column_end** (**Size** i) const
- **column_iterator column_end** (**Size** i)
- const_reverse_column_iterator **column_rbegin** (**Size** i) const
- reverse_column_iterator **column_rbegin** (**Size** i)
- const_reverse_column_iterator **column_rend** (**Size** i) const
- reverse_column_iterator **column_rend** (**Size** i)

Element access

- const_row_iterator **operator[ ]** (**Size**) const
- row_iterator **operator[ ]** (**Size**)
- **Disposable**< **Array** > **diagonal** (void) const

Inspectors

- **Size rows** () const
- **Size columns** () const

Utilities

- void **swap** (**Matrix** &)

Related Functions

(Note that these are not member functions.)

- **Disposable**< **Matrix** > **operator+** (const **Matrix** &, const **Matrix** &)
- **Disposable**< **Matrix** > **operator-** (const **Matrix** &, const **Matrix** &)
- **Disposable**< **Matrix** > **operator** * (const **Matrix** &, double)
- **Disposable**< **Matrix** > **operator** * (double, const **Matrix** &)
- **Disposable**< **Matrix** > **operator/** (const **Matrix** &, double)
- **Disposable**< **Array** > **operator** * (const **Array** &, const **Matrix** &)

- **Disposable**< **Array** > **operator** * (const **Matrix** &, const **Array** &)
- **Disposable**< **Matrix** > **operator** * (const **Matrix** &, const **Matrix** &)
- **Disposable**< **Matrix** > **transpose** (const **Matrix** &)
- **Disposable**< **Matrix** > **outerProduct** (const **Array** &v1, const **Array** &v2)
- template<class Iterator1, class Iterator2> **Disposable**< **Matrix** > **outerProduct** (Iterator1 v1begin, Iterator1 v1end, Iterator2 v2begin, Iterator2 v2end)
- **Disposable**< **Matrix** > **pseudoSqrt** (const **Matrix** &realSymmetricMatrix, SalvagingAlgorithm sa=None)

  returns the pseudo square root of a real symmetric matrix

- **Disposable**< **Matrix** > **matrixSqrt** (const **Matrix** &realSymmetricMatrix)

  returns the pseudo square root of a real symmetric matrix

11.245.2 Member Function Documentation

11.245.2.1 Matrix & operator+= (const Matrix &)

Precondition:
all matrices involved in an algebraic expression must have the same size.

11.245.3 Friends And Related Function Documentation

11.245.3.1 Disposable< Matrix > pseudoSqrt (const Matrix & *realSymmetricMatrix*, SalvagingAlgorithm *sa* = None) [related]

returns the pseudo square root of a real symmetric matrix

returns the pseudo square root of a real symmetric matrix. If the matrix is not positive semi definite, it can returns an approximation of the pseudo square root using a (user selected) salvaging algorithm

For more information see: "The most general methodology to create a valid correlation matrix for risk management and option pricing purposes", by R. Rebonato and P. Jäckel. The Journal of Risk, 2(2), Winter 1999/2000 `http://www.rebonato.com/correlationmatrix.pdf`

Revised and extended in "Monte Carlo Methods in Finance", by Peter Jäckel, Chapter 6

11.245.3.2 Disposable< Matrix > matrixSqrt (const Matrix & *realSymmetricMatrix*) [related]

returns the pseudo square root of a real symmetric matrix

returns the pseudo square root of a real symmetric matrix.

use pseudoSqrt instead

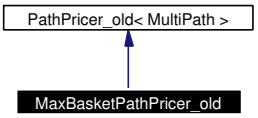
The documentation for this class was generated from the following file:

- ql/Math/**matrix.hpp**

11.246 MaxBasketPathPricer_old Class Reference

`#include <maxbasketpathpricer.hpp>`

Inheritance diagram for MaxBasketPathPricer_old:



11.246.1 Detailed Description

multipath pricer for European-type basket option

The value of the option at expiration is given by the value of the underlying which has best performed.

Public Member Functions

- **MaxBasketPathPricer_old** (const std::vector< double > &underlying, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **MultiPath** &multiPath) const

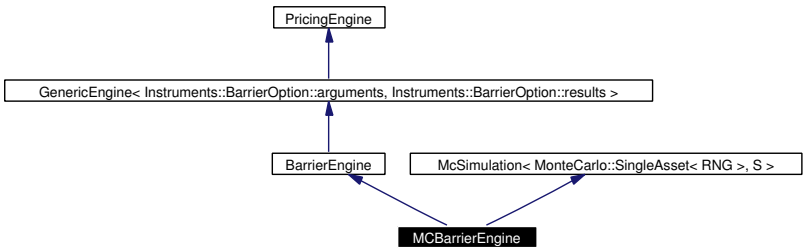
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**maxbasketpathpricer.hpp**
- ql/MonteCarlo/**maxbasketpathpricer.cpp**

11.247 MCBarrierEngine Class Template Reference

`#include <barrierengines.hpp>`

Inheritance diagram for MCBarrierEngine:



11.247.1 Detailed Description

template<class RNG = MonteCarlo::PseudoRandom, class S = Math::Statistics> class QuantLib::PricingEngines::MCBarrierEngine< RNG, S >

Pricing engine for **Barrier**(p. 180) options using Monte Carlo.

Public Member Functions

- **MCBarrierEngine** (**Size** maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, **Size** requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), **Size** maxSamples=**Null**< int >(), bool isBiased=false, long seed=0)
- void **calculate** () const

Protected Types

- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_generator_type path_generator_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_pricer_type path_pricer_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**stats_type stats_type**

Protected Member Functions

- **Handle**< **path_generator_type** > **pathGenerator** () const
- **TimeGrid timeGrid** () const
- **Handle**< **path_pricer_type** > **pathPricer** () const

Protected Attributes

- **Size maxTimeStepsPerYear_**
- **Size requiredSamples_**

- **Size maxSamples_**
- double **requiredTolerance_**
- bool **isBiased_**
- long **seed_**

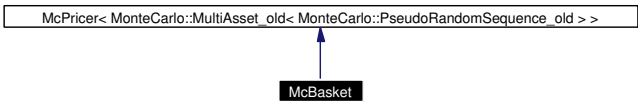
The documentation for this class was generated from the following file:

- ql/PricingEngines/**barrierengines.hpp**

11.248 McBasket Class Reference

`#include <mcbasket.hpp>`

Inheritance diagram for McBasket:



11.248.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

Public Member Functions

- **McBasket** (Option::Type type, const std::vector< double > &underlying, double strike, const **Array** &dividendYield, const **Math::Matrix** &covariance, **Rate** riskFreeRate, double residualTime, bool antitheticVariance, long seed=0)

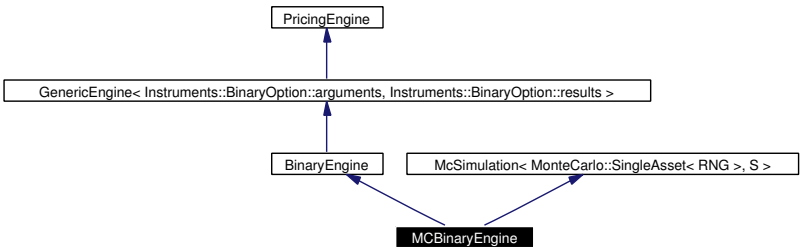
The documentation for this class was generated from the following files:

- ql/Pricers/**mcbasket.hpp**
- ql/Pricers/**mcbasket.cpp**

11.249 MCBinaryEngine Class Template Reference

`#include <binaryengines.hpp>`

Inheritance diagram for MCBinaryEngine:



11.249.1 Detailed Description

template<class RNG = MonteCarlo::PseudoRandom, class S = Math::Statistics> class QuantLib::PricingEngines::MCBinaryEngine< RNG, S >

Pricing engine for **Binary**(p. 195) options using Monte Carlo.

Public Member Functions

- **MCBinaryEngine** (**Size** maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, **Size** requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), **Size** maxSamples=**Null**< int >(), bool isBiased=false, long seed=0)
- void **calculate** () const

Protected Types

- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_generator_type path_generator_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_pricer_type path_pricer_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**stats_type stats_type**
- typedef RNG::ursg_type **my_sequence_type**

Protected Member Functions

- **Handle**< **path_generator_type** > **pathGenerator** () const
- **TimeGrid timeGrid** () const
- **Handle**< **path_pricer_type** > **pathPricer** () const

Protected Attributes

- **Size maxTimeStepsPerYear_**

- **Size requiredSamples_**
- **Size maxSamples_**
- double **requiredTolerance_**
- bool **isBiased_**
- long **seed_**

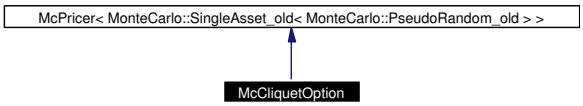
The documentation for this class was generated from the following file:

- ql/PricingEngines/**binaryengines.hpp**

11.250 McCliquetOption Class Reference

`#include <mccliquetoption.hpp>`

Inheritance diagram for McCliquetOption:



11.250.1 Detailed Description

simple example of Monte Carlo pricer

Public Member Functions

- **McCliquetOption** (Option::Type type, double underlying, double moneyness, const std::vector< **Spread** > &dividendYield, const std::vector< **Rate** > &riskFreeRate, const std::vector< **Time** > &times, const std::vector< double > &volatility, double accruedCoupon, double lastFixing, double localCap, double localFloor, double globalCap, double globalFloor, bool redemptionOnly, bool antitheticVariance, long seed=0)

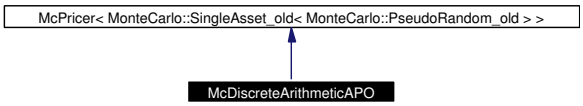
The documentation for this class was generated from the following files:

- ql/Pricers/**mccliquetoption.hpp**
- ql/Pricers/**mccliquetoption.cpp**

11.251 McDiscreteArithmeticAPO Class Reference

`#include <mcdiscretearithmeticapo.hpp>`

Inheritance diagram for McDiscreteArithmeticAPO:



11.251.1 Detailed Description

example of Monte Carlo pricer using a control variate

Todo
Continous Averaging version

Public Member Functions

- **McDiscreteArithmeticAPO** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, const std::vector< **Time** > &times, double volatility, bool antitheticVariance, bool controlVariate, long seed=0)

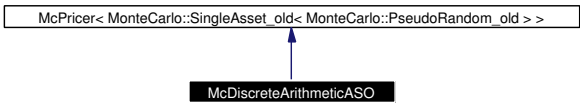
The documentation for this class was generated from the following files:

- ql/Pricers/**mcdiscretearithmeticapo.hpp**
- ql/Pricers/**mcdiscretearithmeticapo.cpp**

11.252 McDiscreteArithmeticASO Class Reference

`#include <mcdiscretearithmeticaso.hpp>`

Inheritance diagram for McDiscreteArithmeticASO:



11.252.1 Detailed Description

example of Monte Carlo pricer using a control variate.

Todo
Continous Averaging version

Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, double underlying, **Spread** dividendYield, **Rate** riskFreeRate, const std::vector< **Time** > &times, double volatility, bool antitheticVariance, bool controlVariate, long seed=0)

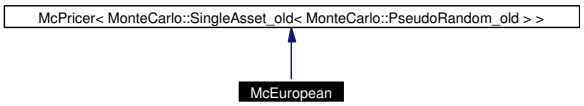
The documentation for this class was generated from the following files:

- ql/Pricers/**mcdiscretearithmeticaso.hpp**
- ql/Pricers/**mcdiscretearithmeticaso.cpp**

11.253 McEuropean Class Reference

`#include <mceuropean.hpp>`

Inheritance diagram for McEuropean:



11.253.1 Detailed Description

simple example of Monte Carlo pricer

Deprecated
use VanillaOption with McEuropeanEngine

Public Member Functions

- **McEuropean** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, double residualTime, double volatility, bool antitheticVariance, long seed=0)

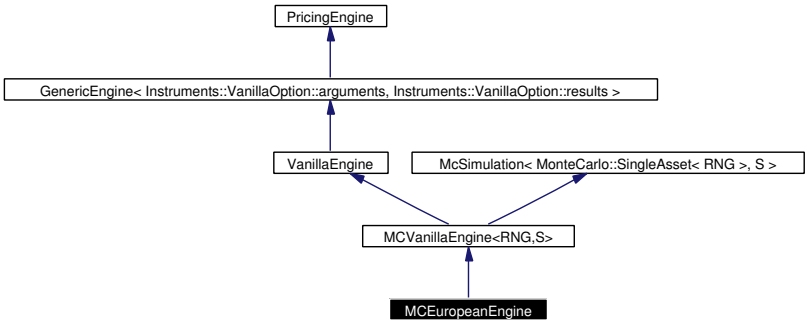
The documentation for this class was generated from the following files:

- ql/Pricers/**mceuropean.hpp**
- ql/Pricers/**mceuropean.cpp**

11.254 MCEuropeanEngine Class Template Reference

`#include <mceuropeanengine.hpp>`

Inheritance diagram for MCEuropeanEngine:



11.254.1 Detailed Description

template<class RNG = MonteCarlo::PseudoRandom, class S = Math::Statistics> class QuantLib::PricingEngines::MCEuropeanEngine< RNG, S >

European option pricing engine using Monte Carlo simulation.

Public Types

- typedef **MCVanillaEngine**< RNG, S >::**path_generator_type path_generator_type**
- typedef **MCVanillaEngine**< RNG, S >::**path_pricer_type path_pricer_type**
- typedef **MCVanillaEngine**< RNG, S >::**stats_type stats_type**

Public Member Functions

- **MCEuropeanEngine** (**Size** maxTimeStepPerYear, bool antitheticVariate=false, bool controlVariate=false, **Size** requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), **Size** maxSamples=**Null**< int >(), long seed=0)

Protected Member Functions

- **TimeGrid timeGrid** () const
- **Handle**< **path_pricer_type** > **pathPricer** () const

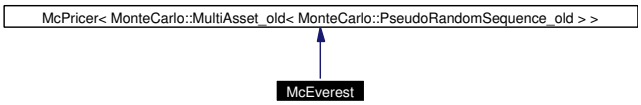
The documentation for this class was generated from the following file:

- ql/PricingEngines/mceuropeanengine.hpp

11.255 McEverest Class Reference

`#include <mceverest.hpp>`

Inheritance diagram for McEverest:



11.255.1 Detailed Description

Everest-type option pricer.

The payoff of an Everest option is simply given by the final price / initial price ratio of the worst performer

Public Member Functions

- **McEverest** (const **Array** &dividendYield, const **Math::Matrix** &covariance, **Rate** riskFreeRate, **Time** residualTime, bool antitheticVariance, long seed=0)

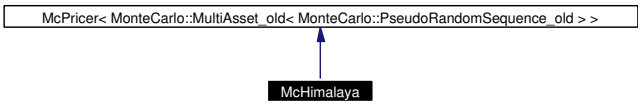
The documentation for this class was generated from the following file:

- ql/Pricers/**mceverest.hpp**

11.256 McHimalaya Class Reference

`#include <mchimalaya.hpp>`

Inheritance diagram for McHimalaya:



11.256.1 Detailed Description

Himalayan-type option pricer.

The payoff of a Himalaya option is computed in the following way: Given a basket of N assets, and N time periods, at end of each period the option who performed the best is added to the average and then discarded from the basket. At the end of the N periods the option pays the max between the strike and the average of the best performers.

Public Member Functions

- **McHimalaya** (const std::vector< double > &underlying, const **Array** &dividendYield, const **Math::Matrix** &covariance, **Rate** riskFreeRate, double strike, const std::vector< **Time** > &times, bool antitheticVariance, long seed=0)

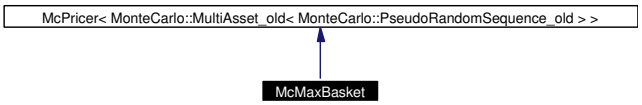
The documentation for this class was generated from the following files:

- ql/Pricers/**mchimalaya.hpp**
- ql/Pricers/**mchimalaya.cpp**

11.257 McMaxBasket Class Reference

`#include <mcmaxbasket.hpp>`

Inheritance diagram for McMaxBasket:



11.257.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

Public Member Functions

- **McMaxBasket** (const std::vector< double > &underlying, const **Array** &dividendYield, const **Math::Matrix** &covariance, **Rate** riskFreeRate, double residualTime, bool antitheticVariance, long seed=0)

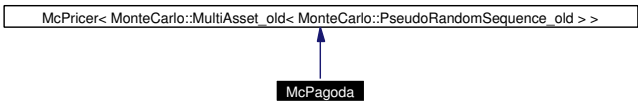
The documentation for this class was generated from the following files:

- ql/Pricers/**mcmaxbasket.hpp**
- ql/Pricers/**mcmaxbasket.cpp**

11.258 McPagoda Class Reference

`#include <mcpagoda.hpp>`

Inheritance diagram for McPagoda:



11.258.1 Detailed Description

roofed Asian option

Given a certain portfolio of assets at the end of the period it is returned the minimum of a given roof and a certain fraction of the positive portfolio performance. If the performance of the portfolio is below then the payoff is null.

Public Member Functions

- **McPagoda** (const std::vector< double > &portfolio, double fraction, double roof, const **Array** &dividendYield, const **Math::Matrix** &covariance, **Rate** riskFreeRate, const std::vector< **Time** > &times, bool antithetic, long seed=0)

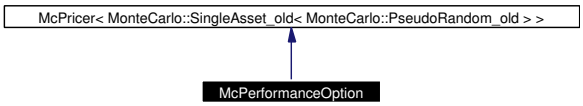
The documentation for this class was generated from the following file:

- ql/Pricers/**mcpagoda.hpp**

11.259 McPerformanceOption Class Reference

`#include <mcperformanceoption.hpp>`

Inheritance diagram for McPerformanceOption:



11.259.1 Detailed Description

Performance option computed using Monte Carlo simulation.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is $ max(S/X- 1) $.

Public Member Functions

- **McPerformanceOption** (Option::Type type, double underlying, double moneyness, const std::vector< **Spread** > &dividendYield, const std::vector< **Rate** > &riskFreeRate, const std::vector< **Time** > &times, const std::vector< double > &volatility, bool antitheticVariance, long seed=0)

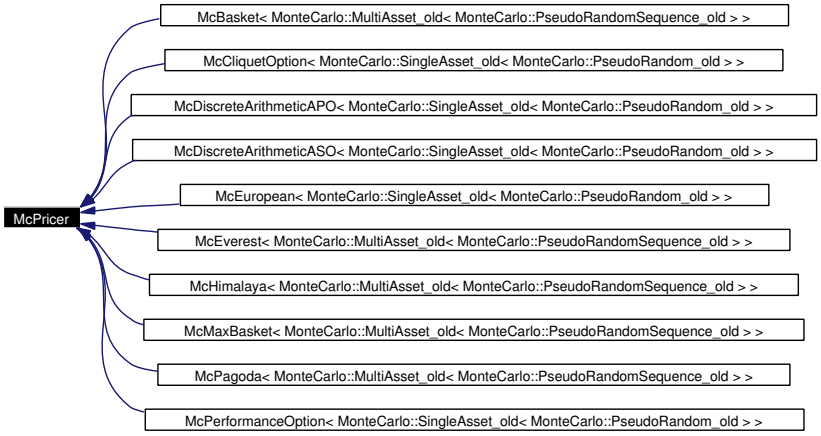
The documentation for this class was generated from the following files:

- ql/Pricers/**mcperformanceoption.hpp**
- ql/Pricers/**mcperformanceoption.cpp**

11.260 McPricer Class Template Reference

`#include <mcpricer.hpp>`

Inheritance diagram for McPricer:



11.260.1 Detailed Description

template<class MC, class S = Math::Statistics> class QuantLib::Pricers::McPricer< MC, S >

base class for Monte Carlo pricers

Eventually this class might be linked to the general tree of pricers, in order to have tools like impliedVolatility available. Also, it could, eventually, offer greeks methods. Deriving a class from **McPricer**(p. 490) gives an easy way to write a Monte Carlo Pricer. See **McEuropean**(p. 483) as example of one factor pricer, Basket as example of multi factor pricer.

Public Member Functions

- double **value** (double tolerance, **Size** maxSample=QL_MAX_INT) const
  add samples until the required tolerance is reached
- double **valueWithSamples** (**Size** samples) const
  simulate a fixed number of samples
- double **errorEstimate** () const
  error Estimated of the samples simulated so far
- const S & **sampleAccumulator** (void) const
  access to the sample accumulator for more statistics

Protected Attributes

- **Handle**< **MonteCarlo::MonteCarloModel**< MC, S > > **mcModel_**

Static Protected Attributes

- const **Size minSample_** = 1023

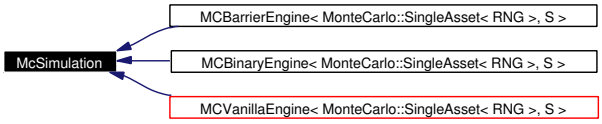
The documentation for this class was generated from the following file:

- ql/Pricers/**mcpricer.hpp**

11.261 McSimulation Class Template Reference

`#include <mcengine.hpp>`

Inheritance diagram for McSimulation:



11.261.1 Detailed Description

template<class MC, class S = Math::Statistics> class QuantLib::PricingEngines::McSimulation< MC, S >

base class for Monte Carlo engines

Eventually this class might offer greeks methods. Deriving a class from McEngine gives an easy way to write a Monte Carlo engine.

See McVanillaEngine as example of one factor engine, McBasketEngine as example of multi factor engine.

Public Types

- typedef **MonteCarlo::MonteCarloModel**< MC, S >::**path_generator_type path_generator_type**
- typedef **MonteCarlo::MonteCarloModel**< MC, S >::**path_pricer_type path_pricer_type**
- typedef **MonteCarlo::MonteCarloModel**< MC, S >::**stats_type stats_type**

Public Member Functions

- double **value** (double tolerance, **Size** maxSample=QL_MAX_INT) const

  add samples until the required tolerance is reached

- double **valueWithSamples** (**Size** samples) const

  simulate a fixed number of samples

- double **errorEstimate** () const

  error estimated using the samples simulated so far

- const **stats_type** & **sampleAccumulator** (void) const

  access to the sample accumulator for richer statistics

Protected Member Functions

- **McSimulation** (bool antitheticVariate, bool controlVariate)
- virtual **Handle**< **path_pricer_type** > **pathPricer** () const =0

- virtual **Handle**< **path_pricer_type** > **controlPathPricer** () const
- virtual **Handle**< **PricingEngine** > **controlPricingEngine** () const
- virtual **Handle**< **path_generator_type** > **pathGenerator** () const =0
- virtual **TimeGrid timeGrid** () const =0

Protected Attributes

- **Handle**< **MonteCarlo::MonteCarloModel**< MC, S > > **mcModel_**
- bool **antitheticVariate_**
- bool **controlVariate_**

Static Protected Attributes

- const **Size minSample_** = 1023

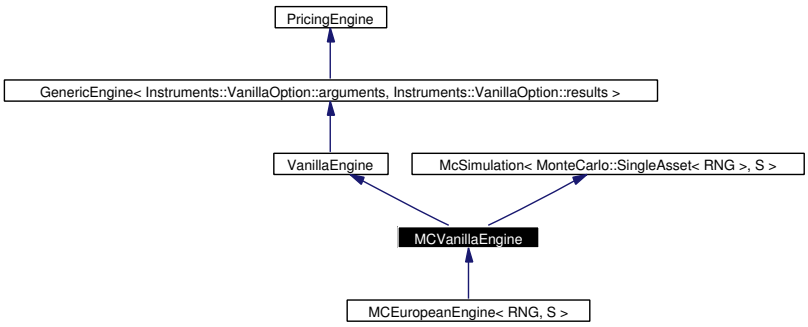
The documentation for this class was generated from the following file:

- ql/PricingEngines/**mcengine.hpp**

11.262 MCVanillaEngine Class Template Reference

`#include <mcengine.hpp>`

Inheritance diagram for MCVanillaEngine:



11.262.1 Detailed Description

template<class RNG = MonteCarlo::PseudoRandom, class S = Math::Statistics> class QuantLib::PricingEngines::MCVanillaEngine< RNG, S >

Base class for Monte Carlo vanilla option engines.

Public Member Functions

- void **calculate** () const

Protected Types

- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_generator_type path_generator_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**path_pricer_type path_pricer_type**
- typedef **McSimulation**< MonteCarlo::SingleAsset< RNG >, S >::**stats_type stats_type**

Protected Member Functions

- **MCVanillaEngine** (**Size** maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, **Size** requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), **Size** maxSamples=**Null**< int >(), long seed=0)
- **Handle**< **path_generator_type** > **pathGenerator** () const

Protected Attributes

- **Size maxTimeStepsPerYear_**
- **Size requiredSamples_**

- **Size maxSamples_**
- double **requiredTolerance_**
- long **seed_**

The documentation for this class was generated from the following file:

- ql/PricingEngines/**mcengine.hpp**

11.263 MersenneTwisterUniformRng Class Reference

`#include <mt19937uniformrng.hpp>`

11.263.1 Detailed Description

Uniform random number generator.

Mersenne Twister random number generator of period 2**19937-1

For more details see `http://www.math.keio.ac.jp/matumoto/emt.html`

Public Types

- typedef **MonteCarlo::Sample**< double > **sample_type**

Public Member Functions

- **MersenneTwisterUniformRng** (unsigned long seed=0)
- **MersenneTwisterUniformRng** (const std::vector< unsigned long > &seeds)
- **sample_type next** () const
- unsigned long **nextInt32** () const

  return a random number on [0,0xffffffff]-interval

11.263.2 Constructor & Destructor Documentation

11.263.2.1 MersenneTwisterUniformRng (unsigned long *seed* = 0) `[explicit]`

if the given seed is 0, a random seed will be chosen based on clock()

11.263.3 Member Function Documentation

11.263.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

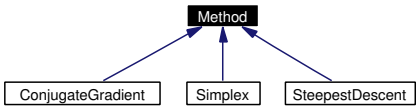
The documentation for this class was generated from the following files:

- ql/RandomNumbers/**mt19937uniformrng.hpp**
- ql/RandomNumbers/**mt19937uniformrng.cpp**

11.264 Method Class Reference

`#include <method.hpp>`

Inheritance diagram for Method:



11.264.1 Detailed Description

Optimization **Method**(p. 497) abstract class for constrained optimization pb.

Public Member Functions

- void **setInitialValue** (const **Array** &initialValue)
  Set initial value.
- void **setEndCriteria** (const **EndCriteria** &endCriteria)
  Set optimization end criteria.
- int & **iterationNumber** () const
  current iteration number
- **EndCriteria** & **endCriteria** () const
  optimization end criteria
- int & **functionEvaluation** () const
  number of evaluation of cost function
- int & **gradientEvaluation** () const
  number of evaluation of cost function gradient
- double & **functionValue** () const
  value of cost function
- double & **gradientNormValue** () const
  value of cost function gradient norm
- **Array** & **x** () const
  current value of the local minimum
- **Array** & **searchDirection** () const
  current value of the search direction
- virtual void **minimize** (const **Problem** &P) const =0
  minimize the optimization problem P

Protected Attributes

- **Array initialValue_**
  initial value of unknowns
- int **iterationNumber_**
  current iteration step in the Optimization process
- **EndCriteria endCriteria_**
  optimization end criteria
- int **functionEvaluation_**
  number of evaluation of cost function and its gradient
- int **gradientEvaluation_**
  number of evaluation of cost function and its gradient
- double **functionValue_**
  function and gradient norm values of the last step
- double **squaredNorm_**
  function and gradient norm values of the last step
- **Array x_**
  current values of the local minimum and the search direction
- **Array searchDirection_**
  current values of the local minimum and the search direction

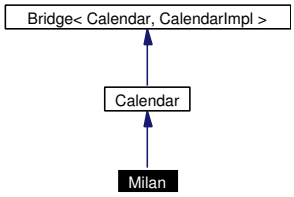
The documentation for this class was generated from the following file:

- ql/Optimization/**method.hpp**

11.265 Milan Class Reference

`#include <milan.hpp>`

Inheritance diagram for Milan:



11.265.1 Detailed Description

Milan calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Easter Monday
- Liberation Day, April 25th
- Labour Day, May 1st
- Republic Day, June 2nd (since 2000)
- Assumption, August 15th
- All Saint's Day, November 1st
- Immaculate Conception, December 8th
- Christmas, December 25th
- St. Stephen, December 26th

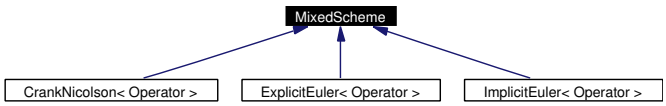
The documentation for this class was generated from the following file:

- ql/Calendars/**milan.hpp**

11.266 MixedScheme Class Template Reference

`#include <mixedscheme.hpp>`

Inheritance diagram for MixedScheme:



11.266.1 Detailed Description

template<class Operator> class QuantLib::FiniteDifferences::MixedScheme< Operator >

Mixed (explicit/implicit) scheme for finite difference methods.

See sect. **The finite differences framework**(p. 31) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:
The differential operator must be linear for this evolver to work.

Todo
add Douglas Scheme

Protected Types

- typedef Operator::arrayType **arrayType**
- typedef Operator **operatorType**
- typedef **BoundaryCondition**< Operator > **bcType**

Protected Member Functions

- **MixedScheme** (const Operator &L, double theta, const std::vector< **Handle**< **bcType** > > &bcs)
- void **step** (arrayType &a, **Time** t)
- void **setStep** (**Time** dt)

Protected Attributes

- Operator **L_**
- Operator **I_**
- Operator **explicitPart_**
- Operator **implicitPart_**
- **Time dt_**
- double **theta_**
- std::vector< **Handle**< **bcType** > > **bcs_**

Friends

- class **FiniteDifferenceModel**< **MixedScheme**< **Operator** > >

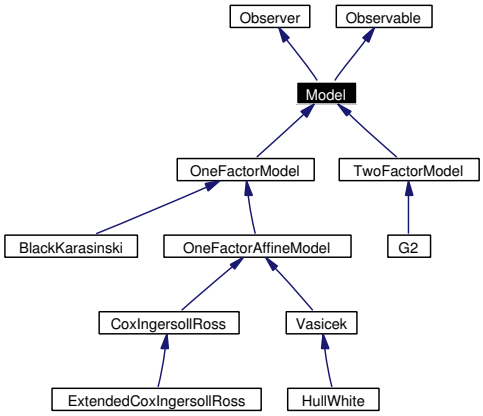
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**mixedscheme.hpp**

11.267 Model Class Reference

`#include <model.hpp>`

Inheritance diagram for Model:



11.267.1 Detailed Description

Abstract short-rate model class.

Public Member Functions

- **Model** (**Size** nArguments)
- void **update** ()
- virtual **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const =0
- void **calibrate** (const std::vector< **Handle**< **CalibrationHelper** > > &instruments, **Optimization::Method** &method, const **Optimization::Constraint** &constraint=**Optimization::Constraint**())

  Calibrate to a set of market instruments (caps/swaptions).

- const **Handle**< **Optimization::Constraint** > & **constraint** () const
- **Disposable**< **Array** > **params** () const

  Returns array of arguments on which calibration is done.

- void **setParams** (const **Array** &params)

Protected Member Functions

- virtual void **generateArguments** ()

Protected Attributes

- std::vector< **Parameter** > **arguments_**
- **Handle**< **Optimization::Constraint** > **constraint_**

Friends

- class **CalibrationFunction**

11.267.2 Member Function Documentation

11.267.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.267.2.2 void calibrate (const std::vector< Handle< CalibrationHelper > > & *instruments*, Optimization::Method & *method*, const Optimization::Constraint & *constraint* = Optimization::Constraint())

Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

The documentation for this class was generated from the following files:

- ql/ShortRateModels/**model.hpp**
- ql/ShortRateModels/**model.cpp**

11.268 MonteCarloModel Class Template Reference

`#include <montecarlomodel.hpp>`

11.268.1 Detailed Description

template<class mc_traits, class stats_traits = Math::Statistics> class QuantLib::MonteCarlo::MonteCarloModel< mc_traits, stats_traits >

General purpose Monte Carlo model for path samples.

The template arguments of this class correspond to available policies for the particular model to be instantiated—i.e., whether it is single- or multi-asset, or whether it should use pseudo-random or low-discrepancy numbers for path generation. Such decisions are grouped in trait classes so as to be orthogonal—see **mctraits.hpp**(p. 923) for examples.

The constructor accepts two safe references, i.e. two smart pointers, one to a path generator and the other to a path pricer. In case of control variate technique the user should provide the additional control option, namely the option path pricer and the option value.

Examples:
DiscreteHedging.cpp.

Public Types

- typedef mc_traits::rsg_type **rsg_type**
- typedef mc_traits::path_generator_type **path_generator_type**
- typedef mc_traits::path_pricer_type **path_pricer_type**
- typedef path_generator_type::sample_type **sample_type**
- typedef path_pricer_type::result_type **result_type**
- typedef stats_traits **stats_type**

Public Member Functions

- **MonteCarloModel** (const **Handle**< path_generator_type > &pathGenerator, const **Handle**< path_pricer_type > &pathPricer, const stats_type &sampleAccumulator, bool antitheticVariate, const **Handle**< path_pricer_type > &cvPathPricer=**Handle**< path_pricer_type >(), result_type cvOptionValue=result_type())
- void **addSamples** (**Size** samples)
- const stats_type & **sampleAccumulator** (void) const

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**montecarlomodel.hpp**

11.269 MoroInverseCumulativeNormal Class Reference

`#include <normaldistribution.hpp>`

11.269.1 Detailed Description

Moro Inverse cumulative normal distribution class.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It uses Beasly and Springer approximation, with an improved approximation for the tails. See Boris Moro, "The Full Monte", 1995, Risk Magazine.

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Peter J. Acklam's approximation is better and is available as **QuantLib::Math::InverseCumulativeNormal**(p. 431)

Public Member Functions

- **MoroInverseCumulativeNormal** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const

The documentation for this class was generated from the following files:

- ql/Math/**normaldistribution.hpp**
- ql/Math/**normaldistribution.cpp**

11.270 MultiPath Class Reference

`#include <multipath.hpp>`

11.270.1 Detailed Description

single random walk

MultiPath(p. 506) contains the list of variations for each asset,

$$\log \frac{Y_{i+1}^j}{Y_i^j} \text{for} i = 0, \ldots, n-1 \qquad \text{and} \qquad j = 0, \ldots, m-1$$

where Y_i^j is the value of the underlying j at discretized time t_i. The first index refers to the underlying, the second to the time position **MultiPath**(p. 506)[j,i]

Todo

make it time-aware

Public Member Functions

- **MultiPath** (**Size** nAsset, const **TimeGrid** &timeGrid)
- **MultiPath** (const std::vector< **Path** > &multiPath)

inspectors

- **Size assetNumber** () const
- **Size pathSize** () const

read/write access to components

- const **Path** & **operator[ ]** (**Size** j) const
- **Path** & **operator[ ]** (**Size** j)

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**multipath.hpp**

11.271 MultiPathGenerator Class Template Reference

`#include <multipathgenerator.hpp>`

11.271.1 Detailed Description

template<class SG> class QuantLib::MonteCarlo::MultiPathGenerator< SG >

Generates a multipath from a random number generator.

MultiPathGenerator<RAG> is a class that returns a random multi path. RAG is a sample generator which returns a random array. It must have the minimal interface:

```
RAG{
    RAG();
    RAG(Matrix& covariance,
        long seed);
    Sample<Array> next();
};
```

Public Types

- typedef **Sample**< **MultiPath** > **sample_type**

Public Member Functions

- **MultiPathGenerator** (const **Array** &drifts, const **Math::Matrix** &covariance, **Time** length, **Size** timeSteps, SG generator)
- **MultiPathGenerator** (const **Array** &drifts, const **Math::Matrix** &covariance, const **TimeGrid** &timeGrid, SG generator)
- const **sample_type** & **next** () const
- const **sample_type** & **antithetic** () const

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**multipathgenerator.hpp**

11.272 MultiPathGenerator_old Class Template Reference

`#include <multipathgenerator.hpp>`

11.272.1 Detailed Description

template<class RAG> class QuantLib::MonteCarlo::MultiPathGenerator_old< RAG >

Generates a multipath from a random number generator.

MultiPathGenerator_old<RAG> is a class that returns a random multi path. RAG is a sample generator which returns a random array. It must have the minimal interface:

```
RAG{
    RAG();
    RAG(Matrix& covariance,
        long seed);
    Sample<Array> next();
};
```

Public Types

- typedef **Sample**< **MultiPath** > **sample_type**

Public Member Functions

- **MultiPathGenerator_old** (const **Array** &drifts, const **Math::Matrix** &covariance, **Time** length, **Size** timeSteps, long seed)
- **MultiPathGenerator_old** (const **Array** &drifts, const **Math::Matrix** &covariance, const **TimeGrid** &times, long seed=0)
- const **sample_type** & **next** () const
- const **sample_type** & **antithetic** () const

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**multipathgenerator.hpp**

11.273 MultivariateAccumulator Class Reference

`#include <multivariateaccumulator.hpp>`

11.273.1 Detailed Description

A sample accumulator for multivariate analysis.

MultivariateAccumulator(p. 509) can accumulate vector-type samples and return the average vector, both in **Array**(p. 173) form and std::vector<double> form, and the covariance matrix

Deprecated
use **SequenceStatistics**(p. 591) instead

Public Member Functions

- **MultivariateAccumulator** (**Size** size)

Inspectors

- **Size size** () const
  size of each sample
- **Size samples** () const
  number of samples collected
- double **weightSum** () const
  sum of data weights
- **Disposable**< **Array** > **mean** () const
  returns the mean as an **Array**(p. 173)
- std::vector< double > **meanVector** () const
  returns the mean as a std::vector<double>
- **Matrix covariance** () const
  returns the covariance **Matrix**(p. 471)
- **Matrix correlation** () const
  returns the correlation **Matrix**(p. 471)

Modifiers

- void **add** (const **Array** &arr, double weight=1.0)
  adds an **Array**(p. 173) *to the collection, possibly with a weight*
- void **add** (const std::vector< double > &vec, double weight=1.0)
  adds a vector<double> to the collection, possibly with a weight
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)

adds a sequence of data to the collection

- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)

  adds a sequence of data to the collection, each with its weight

- void **reset** ()

  resets the data to a null set

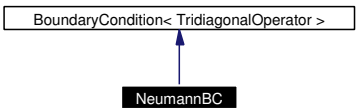
The documentation for this class was generated from the following files:

- ql/Math/**multivariateaccumulator.hpp**
- ql/Math/**multivariateaccumulator.cpp**

11.274 NeumannBC Class Reference

`#include <boundarycondition.hpp>`

Inheritance diagram for NeumannBC:



11.274.1 Detailed Description

Neumann boundary condition (i.e., constant derivative).

Warning:
The value passed must not be the value of the derivative. Instead, it must be comprehensive of the grid step between the first two points–i.e., it must be the difference between f[0] and f[1].

Todo
generalize to time-dependent conditions.

Public Member Functions

- **NeumannBC** (double value, Side side)
- void **applyBeforeApplying** (**TridiagonalOperator** &) const
- void **applyAfterApplying** (**Array** &) const
- void **applyBeforeSolving** (**TridiagonalOperator** &, **Array** &rhs) const
- void **applyAfterSolving** (**Array** &) const
- void **setTime** (**Time** t)

11.274.2 Member Function Documentation

11.274.2.1 void setTime (Time *t*) [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements **BoundaryCondition**< **TridiagonalOperator** > (p. 225).

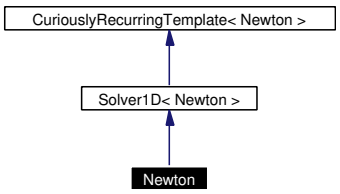
The documentation for this class was generated from the following files:

- ql/FiniteDifferences/**boundarycondition.hpp**
- ql/FiniteDifferences/**boundarycondition.cpp**

11.275 Newton Class Reference

`#include <newton.hpp>`

Inheritance diagram for Newton:



11.275.1 Detailed Description

Newton 1-D solver

Note:
This solver requires that the passed function object implement a method `double derivative(double)`.

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

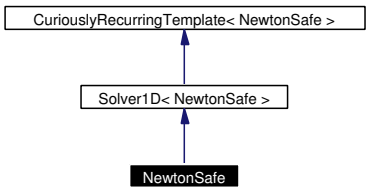
The documentation for this class was generated from the following file:

- ql/Solvers1D/**newton.hpp**

11.276 NewtonSafe Class Reference

`#include <newtonsafe.hpp>`

Inheritance diagram for NewtonSafe:



11.276.1 Detailed Description

safe Newton 1-D solver

Note:
This solver requires that the passed function object implement a method `double derivative(double)`.

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

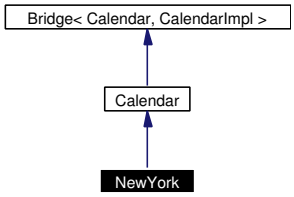
The documentation for this class was generated from the following file:

- ql/Solvers1D/**newtonsafe.hpp**

11.277 NewYork Class Reference

`#include <newyork.hpp>`

Inheritance diagram for NewYork:



11.277.1 Detailed Description

New York calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday, or to Friday if on Saturday)
- Martin Luther King's birthday, third Monday in January
- Washington's birthday, third Monday in February
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veteran's Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

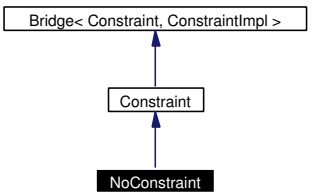
The documentation for this class was generated from the following file:

- ql/Calendars/**newyork.hpp**

11.278 NoConstraint Class Reference

`#include <constraint.hpp>`

Inheritance diagram for NoConstraint:



11.278.1 Detailed Description

No constraint.

The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.279 NonLinearLeastSquare Class Reference

`#include <leastsquare.hpp>`

11.279.1 Detailed Description

Default least square method using a given optimization algorithm (default is conjugate gradient).

min { r(x) : x in R^n }

where r(x) = ||f(x)||^2 the euclidian norm of f(x) for some vector-valued function f from R^n to R^m f = (f1, ..., fm) with fi(x) = bi - phi(x,ti) where bi is the vector of target data and phi is a scalar function.

Assuming the differentiability of f, the gradient of r is define by grad r(x) = f'(x)^t.f(x)

Array(p. 173) vector class has the requirement of the previous class **Handle**(p. 390) class is need to manage pointer to optimization method

Public Member Functions

- **NonLinearLeastSquare** (**Constraint** &c, double accuracy=1e-4, int maxiter=100)
  Default constructor.
- **NonLinearLeastSquare** (**Constraint** &c, double accuracy, int maxiter, **Handle**< **Method** > om)
  Default constructor.
- ~**NonLinearLeastSquare** ()
  Destructor.
- **Array** & **perform** (**LeastSquareProblem** &lsProblem)
  Solve least square problem using numerix solver.
- void **setInitialValue** (const **Array** &initialValue)
- **Array** & **results** ()
  return the results
- double **residualNorm** ()
  return the least square residual norm
- double **lastValue** ()
  return last function value
- int **exitFlag** ()
  return exit flag
- int **iterationsNumber** ()
  return the performed number of iterations

The documentation for this class was generated from the following file:

- ql/Optimization/**leastsquare.hpp**

11.280 NormalDistribution Class Reference

`#include <normaldistribution.hpp>`

11.280.1 Detailed Description

Normal distribution function.

formula here ... Given x it returns its probability in a gaussian normal distribution. It provides the first derivative too.

Public Member Functions

- **NormalDistribution** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const
- double **derivative** (double x) const

The documentation for this class was generated from the following file:

- ql/Math/**normaldistribution.hpp**

11.281 Null Class Template Reference

`#include <null.hpp>`

11.281.1 Detailed Description

template<class Type> class QuantLib::Null< Type >

template class providing a null value for a given type.

Public Member Functions

- **operator Type** () const

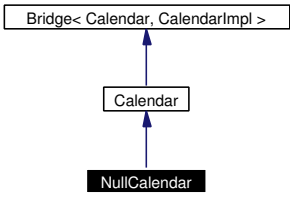
The documentation for this class was generated from the following file:

- ql/**null.hpp**

11.282 NullCalendar Class Reference

`#include <nullcalendar.hpp>`

Inheritance diagram for NullCalendar:



11.282.1 Detailed Description

Calendar(p. 235) for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

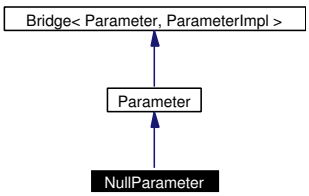
The documentation for this class was generated from the following file:

- ql/Calendars/**nullcalendar.hpp**

11.283 NullParameter Class Reference

`#include <parameter.hpp>`

Inheritance diagram for NullParameter:



11.283.1 Detailed Description

Parameter(p. 539) which is always zero $a(t) = 0$.

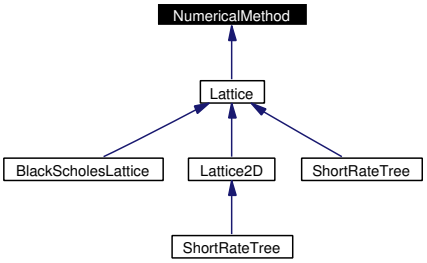
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.284 NumericalMethod Class Reference

`#include <numericalmethod.hpp>`

Inheritance diagram for NumericalMethod:



11.284.1 Detailed Description

Numerical method (Tree, Finite Differences) base class.

Public Member Functions

- **NumericalMethod** (const **TimeGrid** &timeGrid)
- const **TimeGrid** & **timeGrid** () const
- virtual void **initialize** (const **Handle**< **DiscretizedAsset** > &, **Time** time) const =0
- virtual void **rollback** (const **Handle**< **DiscretizedAsset** > &, **Time** to) const =0
- virtual void **rollAlmostBack** (const **Handle**< **DiscretizedAsset** > &, **Time** to) const =0

Protected Attributes

- **TimeGrid t_**

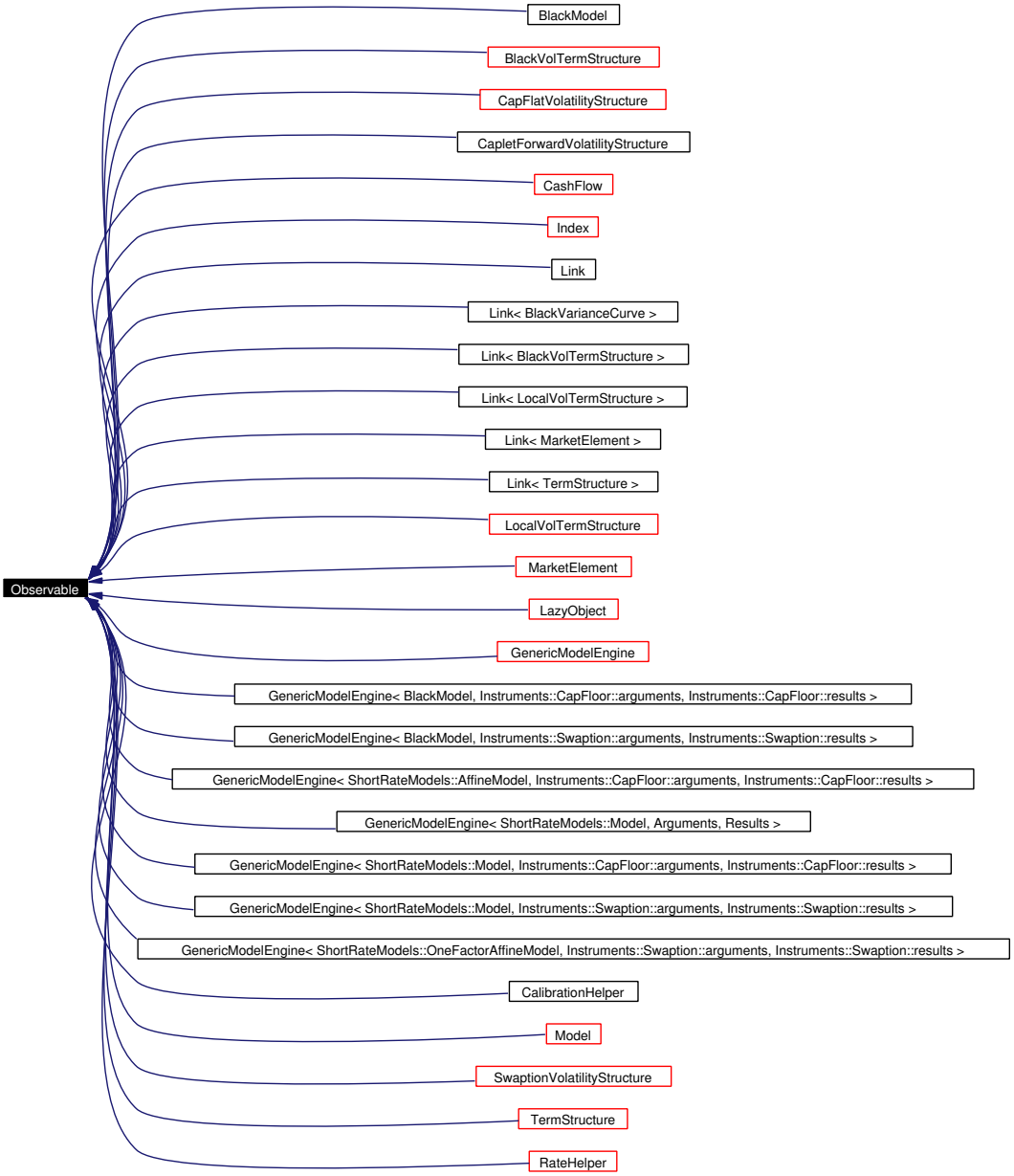
The documentation for this class was generated from the following file:

- ql/**numericalmethod.hpp**

11.285 Observable Class Reference

`#include <observable.hpp>`

Inheritance diagram for Observable:



11.285.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- void **notifyObservers** ()

Friends

- class **Observer**

11.285.2 Member Function Documentation

11.285.2.1 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

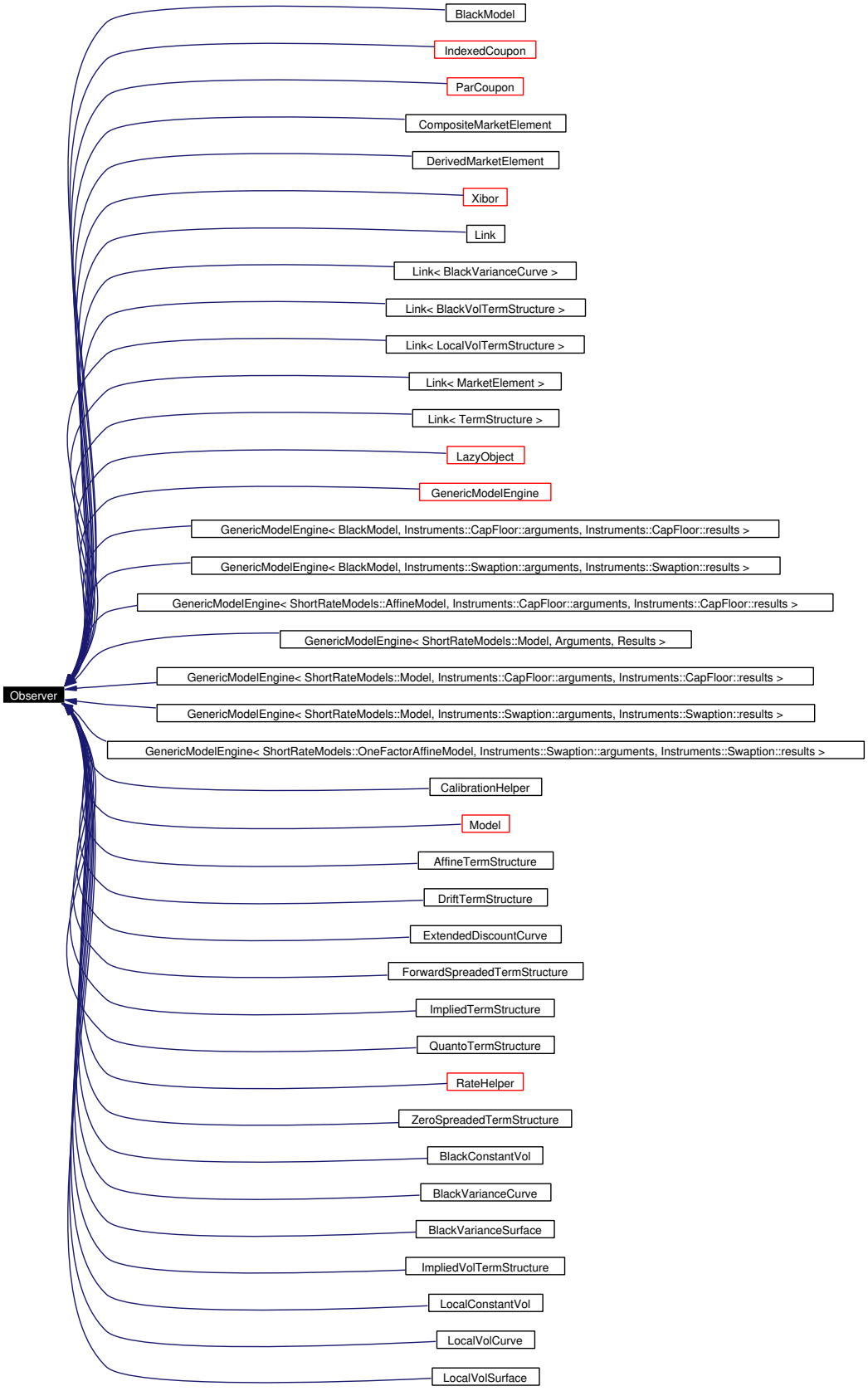
The documentation for this class was generated from the following file:

- ql/Patterns/**observable.hpp**

11.286 Observer Class Reference

`#include <observable.hpp>`

Inheritance diagram for Observer:


Observer
BlackModel
IndexedCoupon
ParCoupon
CompositeMarketElement
DerivedMarketElement
Xibor
Link
Link< BlackVarianceCurve >
Link< BlackVolTermStructure >
Link< LocalVolTermStructure >
Link< MarketElement >
Link< TermStructure >
LazyObject
GenericModelEngine
GenericModelEngine< BlackModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results >
GenericModelEngine< BlackModel, Instruments::Swaption::arguments, Instruments::Swaption::results >
GenericModelEngine< ShortRateModels::AffineModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results >
GenericModelEngine< ShortRateModels::Model, Arguments, Results >
GenericModelEngine< ShortRateModels::Model, Instruments::CapFloor::arguments, Instruments::CapFloor::results >
GenericModelEngine< ShortRateModels::Model, Instruments::Swaption::arguments, Instruments::Swaption::results >
GenericModelEngine< ShortRateModels::OneFactorAffineModel, Instruments::Swaption::arguments, Instruments::Swaption::results >
CalibrationHelper
Model
AffineTermStructure
DriftTermStructure
ExtendedDiscountCurve
ForwardSpreadedTermStructure
ImpliedTermStructure
QuantoTermStructure
RateHelper
ZeroSpreadedTermStructure
BlackConstantVol
BlackVarianceCurve
BlackVarianceSurface
ImpliedVolTermStructure
LocalConstantVol
LocalVolCurve
LocalVolSurface

11.286.1 Detailed Description

Object that gets notified when a given observable changes.

Public Member Functions

- **Observer** (const **Observer** &)
- **Observer** & **operator=** (const **Observer** &)
- void **registerWith** (const **Handle**< **Observable** > &)
- void **unregisterWith** (const **Handle**< **Observable** > &)
- virtual void **update** ()=0

11.286.2 Member Function Documentation

11.286.2.1 virtual void update () [pure virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in **BlackModel** (p. 211), **IndexedCoupon** (p. 420), **ParCoupon** (p. 542), **Xibor** (p. 676), **DerivedMarketElement** (p. 297), **CompositeMarketElement** (p. 267), **LazyObject** (p. 444), **GenericModelEngine** (p. 382), **LatticeShortRateModelEngine** (p. 442), **Link** (p. 455), **CalibrationHelper** (p. 241), **Model** (p. 503), **AffineTermStructure** (p. 160), **DriftTermStructure** (p. 319), **ExtendedDiscountCurve** (p. 341), **ForwardSpreadedTermStructure** (p. 364), **ImpliedTermStructure** (p. 410), **QuantoTermStructure** (p. 571), **RateHelper** (p. 578), **ZeroSpreadedTermStructure** (p. 682), **BlackConstantVol** (p. 208), **BlackVarianceCurve** (p. 217), **BlackVarianceSurface** (p. 219), **ImpliedVolTermStructure** (p. 412), **LocalConstantVol** (p. 458), **LocalVolCurve** (p. 460), **LocalVolSurface** (p. 462), **GenericModelEngine**< **ShortRateModels::OneFactorAffineModel, Instruments::Swaption::arguments, Instruments::Swaption::results** > (p. 382), **GenericModelEngine**< **BlackModel, Instruments::Swaption::arguments, Instruments::Swaption::results** > (p. 382), **GenericModelEngine**< **ShortRateModels::Model, Arguments, Results** > (p. 382), **GenericModelEngine**< **BlackModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results** > (p. 382), **GenericModelEngine**< **ShortRateModels::AffineModel, Instruments::CapFloor::arguments, Instruments::CapFloor::results** > (p. 382), **GenericModelEngine**< **ShortRateModels::Model, Instruments::CapFloor::arguments, Instruments::CapFloor::results** > (p. 382), **GenericModelEngine**< **ShortRateModels::Model, Instruments::Swaption::arguments, Instruments::Swaption::results** > (p. 382), **LatticeShortRateModelEngine**< **Instruments::Swaption::arguments, Instruments::Swaption::results** > (p. 442), **LatticeShortRateModelEngine**< **Instruments::CapFloor::arguments, Instruments::CapFloor::results** > (p. 442), **Link**< **LocalVolTermStructure** > (p. 455), **Link**< **BlackVarianceCurve** > (p. 455), **Link**< **MarketElement** > (p. 455), **Link**< **BlackVolTermStructure** > (p. 455), and **Link**< **TermStructure** > (p. 455).

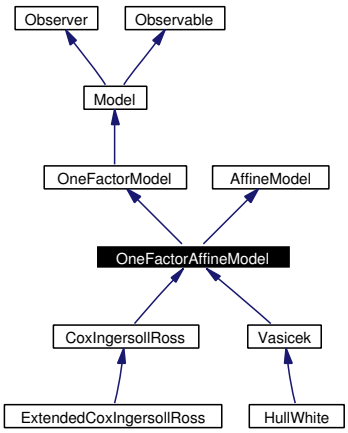
The documentation for this class was generated from the following file:

- ql/Patterns/**observable.hpp**

11.287 OneFactorAffineModel Class Reference

`#include <onefactormodel.hpp>`

Inheritance diagram for OneFactorAffineModel:



11.287.1 Detailed Description

Single-factor affine base class.

Single-factor models with an analytical formula for discount bonds should inherit from this class. They must then implement the functions $A(t,T)$ and $B(t,T)$ such that

$$P(t,T,r_t) = A(t,T)e^{-B(t,T)r_t}.$$

Public Member Functions

- **OneFactorAffineModel** (**Size** nArguments)
- double **discountBond** (**Time** now, **Time** maturity, **Rate** rate) const
- **DiscountFactor discount** (**Time** t) const

  Implied discount curve.

Protected Member Functions

- virtual double **A** (**Time** t, **Time** T) const =0
- virtual double **B** (**Time** t, **Time** T) const =0

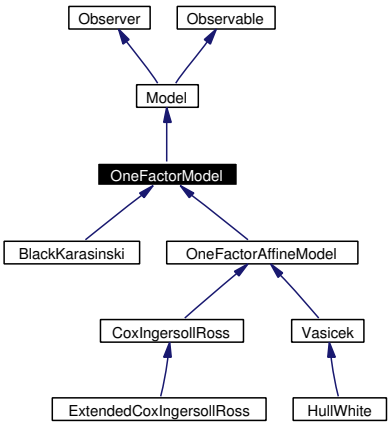
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**onefactormodel.hpp**

11.288 OneFactorModel Class Reference

`#include <onefactormodel.hpp>`

Inheritance diagram for OneFactorModel:



11.288.1 Detailed Description

Single-factor short-rate model abstract class.

Public Member Functions

- **OneFactorModel** (**Size** nArguments)
- virtual **Handle**< **ShortRateDynamics** > **dynamics** () const =0

  returns the short-rate dynamics

- virtual **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Return by default a trinomial recombining tree.

The documentation for this class was generated from the following files:

- ql/ShortRateModels/**onefactormodel.hpp**
- ql/ShortRateModels/**onefactormodel.cpp**

11.289 OneFactorModel::ShortRateDynamics Class Reference

`#include <onefactormodel.hpp>`

11.289.1 Detailed Description

Base class describing the short-rate dynamics.

Public Member Functions

- **ShortRateDynamics** (const **Handle**< **DiffusionProcess** > &process)
- virtual double **variable** (**Time** t, **Rate** r) const =0

  Compute state variable from short rate.

- virtual **Rate shortRate** (**Time** t, double variable) const =0

  Compute short rate from state variable.

- const **Handle**< **DiffusionProcess** > & **process** ()

  Returns the risk-neutral dynamics of the state variable.

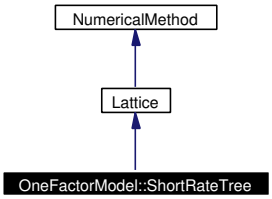
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**onefactormodel.hpp**

11.290 OneFactorModel::ShortRateTree Class Reference

`#include <onefactormodel.hpp>`

Inheritance diagram for OneFactorModel::ShortRateTree:



11.290.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

Public Member Functions

- **ShortRateTree** (const **Handle**< **Lattices::Tree** > &tree, const **Handle**< **ShortRateDynamics** > &dynamics, const **TimeGrid** &timeGrid)

  Plain tree build-up from short-rate dynamics.

- **ShortRateTree** (const **Handle**< **Lattices::Tree** > &tree, const **Handle**< **ShortRateDynamics** > &dynamics, const **Handle**< TermStructureFittingParameter::NumericalImpl > &phi, const **TimeGrid** &timeGrid)

  Tree build-up + numerical fitting to term-structure.

- **Size size** (**Size** i) const
- **DiscountFactor discount** (**Size** i, **Size** index) const

  Discount factor at time t_i and node indexed by index.

Protected Member Functions

- **Size descendant** (**Size** i, **Size** index, **Size** branch) const

  Tree properties.

- double **probability** (**Size** i, **Size** index, **Size** branch) const

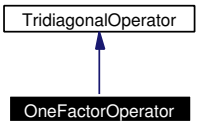
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**onefactormodel.hpp**

11.291 OneFactorOperator Class Reference

`#include <onefactoroperator.hpp>`

Inheritance diagram for OneFactorOperator:



11.291.1 Detailed Description

Interest-rate single factor model differential operator.

Public Member Functions

- **OneFactorOperator** (const **Array** &grid, const **Handle**< **ShortRateModels::OneFactorModel::ShortRateDynamics** > &process)

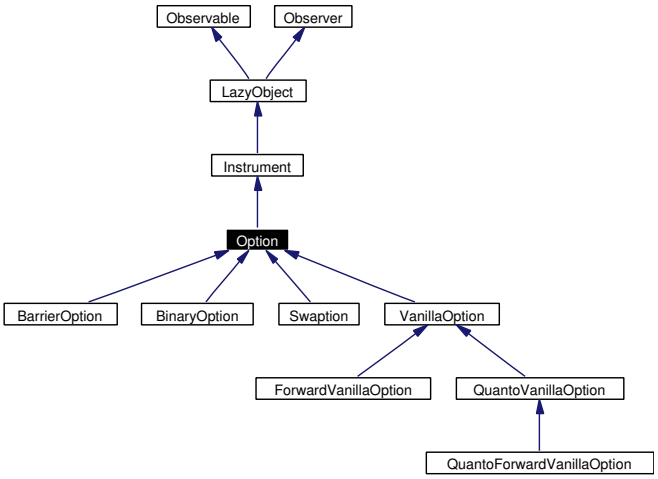
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**onefactoroperator.hpp**

11.292 Option Class Reference

`#include <option.hpp>`

Inheritance diagram for Option:



11.292.1 Detailed Description

base option class

Examples:
DiscreteHedging.cpp.

Public Types

- enum **Type** { **Call**, **Put**, **Straddle** }

Public Member Functions

- **Option** (const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >(), const std::string &isinCode="", const std::string &description="")

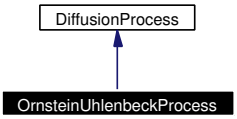
The documentation for this class was generated from the following file:

- ql/**option.hpp**

11.293 OrnsteinUhlenbeckProcess Class Reference

`#include <diffusionprocess.hpp>`

Inheritance diagram for OrnsteinUhlenbeckProcess:



11.293.1 Detailed Description

Ornstein-Uhlenbeck process class.

This class describes the Ornstein-Uhlenbeck process governed by

$$dx = -ax_t dt + \sigma dW_t.$$

Public Member Functions

- **OrnsteinUhlenbeckProcess** (double speed, double vol, double x0=0.0)
- double **drift** (**Time** t, double x) const

  returns the drift part of the equation, i.e. $\mu(t, x_t)$
- double **diffusion** (**Time** t, double x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- double **expectation** (**Time** t0, double x0, **Time** dt) const

  returns the expectation of the process after a time interval
- double **variance** (**Time** t0, double x0, **Time** dt) const

  returns the variance of the process after a time interval

11.293.2 Member Function Documentation

11.293.2.1 double expectation (Time *t0*, double *x0*, Time *dt*) const [virtual]

returns the expectation of the process after a time interval

returns $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $x_0 + \mu(t_0, x_0)\Delta t$.

Reimplemented from **DiffusionProcess** (p. 298).

11.293.2.2 double variance (Time *t0*, double *x0*, Time *dt*) const [virtual]

returns the variance of the process after a time interval

returns $Var(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $\sigma(t_0, x_0)^2\Delta t$.

Reimplemented from **DiffusionProcess** (p. 298).

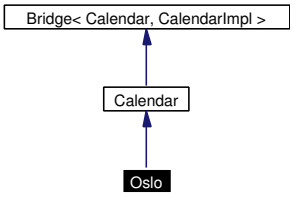
The documentation for this class was generated from the following file:

- ql/**diffusionprocess.hpp**

11.294 Oslo Class Reference

`#include <oslo.hpp>`

Inheritance diagram for Oslo:



11.294.1 Detailed Description

Oslo calendar

Holidays:

- Saturdays
- Sundays
- Holy Thursday
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- May Day, May 1st
- National Independence Day, May 17st
- Christmas, December 25th
- Boxing Day, December 26th

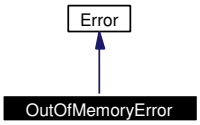
The documentation for this class was generated from the following file:

- ql/Calendars/**oslo.hpp**

11.295 OutOfMemoryError Class Reference

`#include <errors.hpp>`

Inheritance diagram for OutOfMemoryError:



11.295.1 Detailed Description

Specialized error.

Thrown upon failed allocation.

Public Member Functions

- **OutOfMemoryError** (const std::string &whatClass="unknown class")

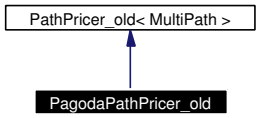
The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.296 PagodaPathPricer_old Class Reference

`#include <pagodapathpricer.hpp>`

Inheritance diagram for PagodaPathPricer_old:



11.296.1 Detailed Description

multipath pricer for pagoda options

A pagoda option is a multi-asset asian option with a cap (the pagoda "roof"). Given a portfolio of assets the payoff is the arithmetic average of the portfolio performance, with a maximum cap given by the roof.

Public Member Functions

- **PagodaPathPricer_old** (const std::vector< double > &underlying, double roof, **DiscountFactor** discount, bool useAntitheticVariance)
- double **operator()** (const **MultiPath** &path) const

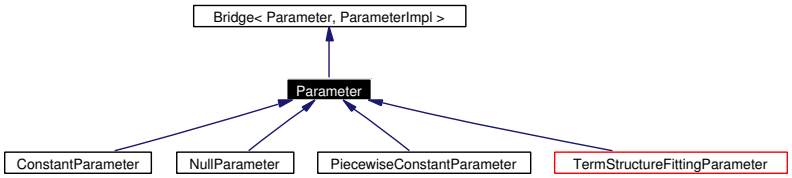
The documentation for this class was generated from the following files:

- ql/MonteCarlo/**pagodapathpricer.hpp**
- ql/MonteCarlo/**pagodapathpricer.cpp**

11.297 Parameter Class Reference

`#include <parameter.hpp>`

Inheritance diagram for Parameter:



11.297.1 Detailed Description

Base class for model arguments.

Public Member Functions

- const **Array** & **params** () const
- void **setParam** (**Size** i, double x)
- bool **testParams** (const **Array** &params) const
- **Size size** () const
- double **operator()** (**Time** t) const
- const **Handle**< **ParameterImpl** > & **implementation** () const

Protected Member Functions

- **Parameter** (**Size** size, const **Handle**< **ParameterImpl** > &impl, const **Optimization::Constraint** &constraint)

Protected Attributes

- **Array params_**
- **Optimization::Constraint constraint_**

The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.298 ParameterImpl Class Reference

`#include <parameter.hpp>`

11.298.1 Detailed Description

Base class for model parameter implementation.

Public Member Functions

- virtual double **value** (const **Array** &params, **Time** t) const =0

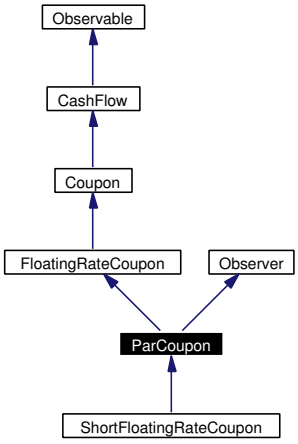
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.299 ParCoupon Class Reference

`#include <parcoupon.hpp>`

Inheritance diagram for ParCoupon:



11.299.1 Detailed Description

coupon at par on a term structure

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ParCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

CashFlow interface

- double **amount** () const
  returns the amount of the cash flow

Coupon interface

- **DayCounter dayCounter** () const
  day counter for accrual calculation

FloatingRateCoupon interface

- **Rate fixing** () const

- **Date fixingDate** () const

Inspectors

- const **Handle**< **Indexes::Xibor** > & **index** () const

Observer interface

- void **update** ()

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.299.2 Member Function Documentation

11.299.2.1 double amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. 252).

Reimplemented in **ShortFloatingRateCoupon** (p. 594).

11.299.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

The documentation for this class was generated from the following files:

- ql/CashFlows/**parcoupon.hpp**
- ql/CashFlows/**parcoupon.cpp**

11.300 Path Class Reference

`#include <path.hpp>`

11.300.1 Detailed Description

single factor random walk

Examples:
DiscreteHedging.cpp.

Public Member Functions

- **Path** (const **TimeGrid** &timeGrid, const **Array** &drift=**Array**(), const **Array** &diffusion=**Array**())

inspectors

- double **operator[ ]** (int i) const
- **Size size** () const

read/write access to components

- const **TimeGrid** & **timeGrid** () const
- **TimeGrid** & **timeGrid** ()
- const **Array** & **drift** () const
- **Array** & **drift** ()
- const **Array** & **diffusion** () const
- **Array** & **diffusion** ()

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**path.hpp**

11.301 PathGenerator Class Template Reference

`#include <pathgenerator.hpp>`

11.301.1 Detailed Description

template<class SG> class QuantLib::MonteCarlo::PathGenerator< SG >

Generates random paths using a sequence generator.

Generates random paths with drift(S,t) and variance(S,t) using a sequence generator

Public Types

- typedef **Sample**< **Path** > **sample_type**

Public Member Functions

- **PathGenerator** (const **Handle**< **DiffusionProcess** > &diffProcess, **Time** length, **Size** timeSteps, const SG &generator)
- **PathGenerator** (const **Handle**< **DiffusionProcess** > &diffProcess, const **TimeGrid** &timeGrid, const SG &generator)

inspectors

- const **sample_type** & **next** () const
- const **sample_type** & **antithetic** () const
- **Size size** () const
- const **TimeGrid** & **timeGrid** () const

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**pathgenerator.hpp**

11.302 PathGenerator_old Class Template Reference

`#include <pathgenerator.hpp>`

11.302.1 Detailed Description

template<class RNG> class QuantLib::MonteCarlo::PathGenerator_old< RNG >

Generates random paths from a random number generator.

Deprecated
use **PathGenerator**(p. 544) instead

Examples:
DiscreteHedging.cpp.

Public Types

- typedef **Sample**< **Path** > **sample_type**

Public Member Functions

- **PathGenerator_old** (double drift, double variance, **Time** length, **Size** timeSteps, long seed=0)
- **PathGenerator_old** (double drift, double variance, const **TimeGrid** &times, long seed=0)
- **PathGenerator_old** (const std::vector< double > &drift, const std::vector< double > &variance, const **TimeGrid** &times, long seed=0)

inspectors

- const **sample_type** & **next** () const
- const **sample_type** & **antithetic** () const
- **Size size** () const

11.302.2 Constructor & Destructor Documentation

11.302.2.1 PathGenerator_old (double *drift*, double *variance*, const TimeGrid & *times*, long *seed* = 0)

Warning:
the initial time is assumed to be zero and must **not** be included in the passed vector

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**pathgenerator.hpp**

11.303 PathPricer Class Template Reference

`#include <pathpricer.hpp>`

11.303.1 Detailed Description

template<class PathType, class ValueType = double> class QuantLib::MonteCarlo::PathPricer< PathType, ValueType >

base class for path pricers

Given a path the value of an option is returned on that path.

Public Member Functions

- **PathPricer** (const **RelinkableHandle**< **TermStructure** > &riskFreeTS)
- virtual ValueType **operator()** (const PathType &path) const =0

Protected Attributes

- **RelinkableHandle**< **TermStructure** > **riskFreeTS_**

The documentation for this class was generated from the following file:

- ql/MonteCarlo/**pathpricer.hpp**

11.304 PathPricer_old Class Template Reference

`#include <pathpricer.hpp>`

11.304.1 Detailed Description

template<class PathType, class ValueType = double> class QuantLib::MonteCarlo::PathPricer_old< PathType, ValueType >

base class for path pricers

Given a path the value of an option is returned on that path.

Deprecated
use **PathPricer**(p. 546) instead

Examples:
DiscreteHedging.cpp.

Public Member Functions

- **PathPricer_old** (**DiscountFactor** discount, bool useAntitheticVariance)
- virtual ValueType **operator()** (const PathType &path) const =0

Protected Attributes

- **DiscountFactor discount_**
- bool **useAntitheticVariance_**

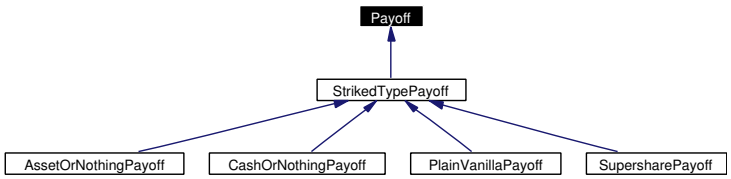
The documentation for this class was generated from the following file:

- ql/MonteCarlo/**pathpricer.hpp**

11.305 Payoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for Payoff:



11.305.1 Detailed Description

Option(p. 533) payoff base class.

Public Member Functions

- virtual double **operator()** (double price) const =0

The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.306 PerformanceOption Class Reference

`#include <performanceoption.hpp>`

11.306.1 Detailed Description

Performance option.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is $ max(S/X- 1) $.

Public Member Functions

- **PerformanceOption** (Option::Type type, double underlying, double moneyness, const std::vector< **Spread** > &dividendYield, const std::vector< **Rate** > &riskFreeRate, const std::vector< **Time** > &times, const std::vector< double > &volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const

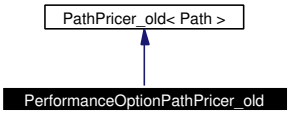
The documentation for this class was generated from the following files:

- ql/Pricers/**performanceoption.hpp**
- ql/Pricers/**performanceoption.cpp**

11.307 PerformanceOptionPathPricer_old Class Reference

`#include <performanceoptionpathpricer.hpp>`

Inheritance diagram for PerformanceOptionPathPricer_old:



11.307.1 Detailed Description

path pricer for performance options

Public Member Functions

- **PerformanceOptionPathPricer_old** (Option::Type type, double underlying, double moneyness, const std::vector< **DiscountFactor** > &discounts, bool useAntitheticVariance)
- double **operator()** (const **Path** &path) const

The documentation for this class was generated from the following files:

- ql/MonteCarlo/**performanceoptionpathpricer.hpp**
- ql/MonteCarlo/**performanceoptionpathpricer.cpp**

11.308 Period Class Reference

`#include <date.hpp>`

11.308.1 Detailed Description

Time period described by a number of a given time unit.

Public Member Functions

- **Period** (int n, **TimeUnit** units)
- int **length** () const
- **TimeUnit units** () const

Related Functions

(Note that these are not member functions.)

- bool **operator==** (const **Period** &, const **Period** &)
- bool **operator**< (const **Period** &, const **Period** &)

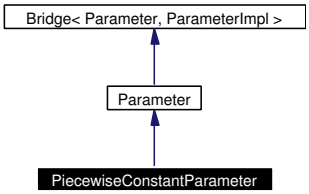
The documentation for this class was generated from the following file:

- ql/**date.hpp**

11.309 PiecewiseConstantParameter Class Reference

`#include <parameter.hpp>`

Inheritance diagram for PiecewiseConstantParameter:



11.309.1 Detailed Description

Piecewise constant parameter.

$a(t) = a_i if t_{i-1} \geq t < t_i$. This kind of parameter is usually used to enhance the fitting of a model

Public Member Functions

- **PiecewiseConstantParameter** (const std::vector< **Time** > &times)

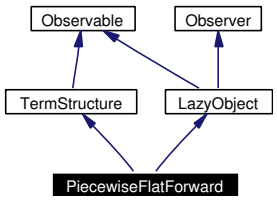
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.310 PiecewiseFlatForward Class Reference

`#include <piecewiseflatforward.hpp>`

Inheritance diagram for PiecewiseFlatForward:



11.310.1 Detailed Description

Piecewise flat forward term structure.

This term structure is bootstrapped on a number of interest rate instruments which are passed as a vector of handles to **RateHelper**(p. 577) instances. Their maturities mark the boundaries of the flat forward segments.

The values of the forward rates for each segment are determined sequentially starting from the earliest period to the latest.

The value for each segment is chosen so that the instrument whose maturity marks the end of such segment is correctly repriced on the curve.

Rates are assumed to be annual continuos compounding.

Warning:
The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

Public Member Functions

- **PiecewiseFlatForward** (const **Date** &todaysDate, const **Date** &referenceDate, const std::vector< **Handle**< **RateHelper** > > &instruments, const **DayCounter** &dayCounter, double accuracy=1.0e-12)
- **PiecewiseFlatForward** (const **Date** &todaysDate, const std::vector< **Date** > &dates, const std::vector< **Rate** > &forwards, const **DayCounter** &dayCounter)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date todaysDate** () const
  today's date
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1

- const std::vector< **Date** > & **dates** () const
- **Date maxDate** () const

  the latest date for which the curve can return rates

- const std::vector< **Time** > & **times** () const
- **Time maxTime** () const

  the latest time for which the curve can return rates

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const

  zero-yield calculation

- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const

  discount calculation

- **Rate forwardImpl** (**Time**, bool extrapolate=false) const

  instantaneous forward-rate calculation

- **Rate compoundForwardImpl** (**Time** t, int compFreq, bool extrapolate) const

  compound forward-rate calculation

Friends

- class **FFObjFunction**

11.310.2 Constructor & Destructor Documentation

11.310.2.1 PiecewiseFlatForward (const Date & *todaysDate*, const std::vector< Date > & *dates*, const std::vector< Rate > & *forwards*, const DayCounter & *dayCounter*)

In this constructor, the first date must be the reference date of the curve, the other dates are the nodes of the term structure. The forward rate at index i is used in the period $t_{i-1} < t \leq t_i$. Therefore, forwards[0] is used only to compute the zero yield for $t = 0$.

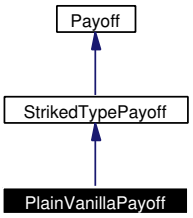
The documentation for this class was generated from the following files:

- ql/TermStructures/**piecewiseflatforward.hpp**
- ql/TermStructures/**piecewiseflatforward.cpp**

11.311 PlainVanillaPayoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for PlainVanillaPayoff:



11.311.1 Detailed Description

The former PlainPayoff has been renamed **PlainVanillaPayoff**(p. 555) to stress that fact that now nobody elses derives from it

Examples:
DiscreteHedging.cpp.

Public Member Functions

- **PlainVanillaPayoff** (Option::Type type, double strike)
- double **operator()** (double price) const

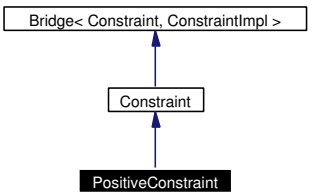
The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.312 PositiveConstraint Class Reference

`#include <constraint.hpp>`

Inheritance diagram for PositiveConstraint:



11.312.1 Detailed Description

Constraint(p. 270) imposing positivity to all arguments.

The documentation for this class was generated from the following file:

- ql/Optimization/**constraint.hpp**

11.313 PostconditionNotSatisfiedError Class Reference

`#include <errors.hpp>`

Inheritance diagram for PostconditionNotSatisfiedError:

11.313.1 Detailed Description

Specialized error.

Thrown upon an unsatisfied postcondition.

Public Member Functions

- **PostconditionNotSatisfiedError** (const std::string &what="")

The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.314 PreconditionNotSatisfiedError Class Reference

`#include <errors.hpp>`

Inheritance diagram for PreconditionNotSatisfiedError:

11.314.1 Detailed Description

Specialized error.

Thrown upon an unsatisfied precondition.

Public Member Functions

- **PreconditionNotSatisfiedError** (const std::string &what="")

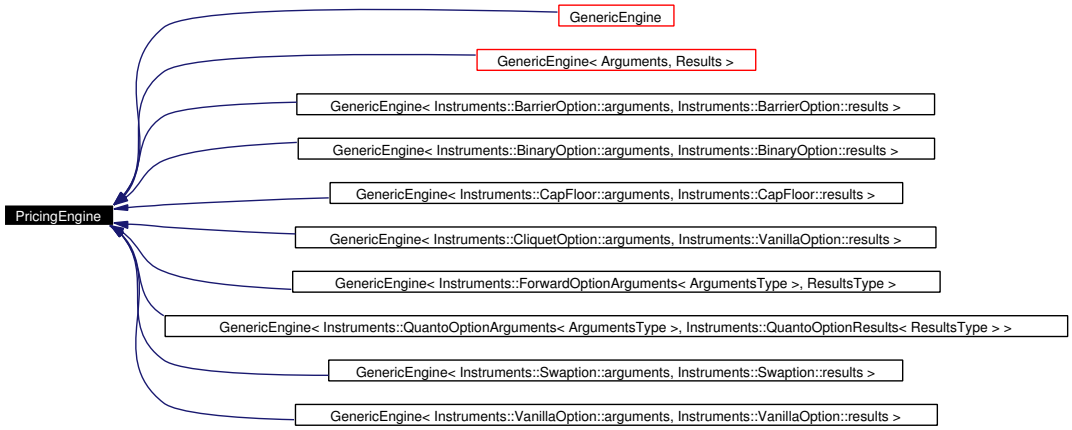
The documentation for this class was generated from the following file:

- ql/**errors.hpp**

11.315 PricingEngine Class Reference

`#include <pricingengine.hpp>`

Inheritance diagram for PricingEngine:



11.315.1 Detailed Description

base class for pricing engines

Public Member Functions

- virtual **Arguments** * **arguments** () const =0
- virtual const **Results** * **results** () const =0
- virtual void **reset** () const =0
- virtual void **calculate** () const =0

The documentation for this class was generated from the following file:

- ql/**pricingengine.hpp**

11.316 PrimeNumbers Class Reference

`#include <primenumbers.hpp>`

11.316.1 Detailed Description

Prime numbers calculator.

Prime numbers calculator. Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

Static Public Member Functions

- unsigned long **get** (**Size** absoluteIndex)
  Get and store one after another.

The documentation for this class was generated from the following files:

- ql/Math/**primenumbers.hpp**
- ql/Math/**primenumbers.cpp**

11.317 Problem Class Reference

`#include <problem.hpp>`

11.317.1 Detailed Description

Constrained optimization problem.

Public Member Functions

- **Problem** (**CostFunction** &f, **Constraint** &c, **Method** &meth)
  default constructor
- double **value** (const **Array** &x) const
  call cost function computation and increment evaluation counter
- void **gradient** (**Array** &grad_f, const **Array** &x) const
  call cost function gradient computation and increment
- double **valueAndGradient** (**Array** &grad_f, const **Array** &x) const
  call cost function computation and it gradient
- **Method** & **method** () const
  Constrained optimization method.
- **Constraint** & **constraint** () const
  Constraint(p. 270).
- **CostFunction** & **costFunction** () const
  Cost function.
- void **minimize** () const
  Minimization.
- **Array** & **minimumValue** () const

Protected Attributes

- **CostFunction** & **costFunction_**
  Unconstrained cost function.
- **Constraint** & **constraint_**
  Constraint(p. 270).
- **Method** & **method_**
  constrained optimization method

The documentation for this class was generated from the following file:

- ql/Optimization/**problem.hpp**

11.318 processing_iterator Class Template Reference

`#include <processingiterator.hpp>`

11.318.1 Detailed Description

template<class Iterator, class UnaryFunction> class QuantLib::Utilities::processing_iterator< Iterator, UnaryFunction >

Iterator mapping a unary function to an underlying sequence.

This iterator advances an underlying iterator and returns the values obtained by applying a unary function to the values such iterator points to.

This class was implemented based on Christopher Baus and Thomas Becker, *Custom Iterators for the STL*, included in the proceedings of the First Workshop on C++ Template Programming, Erfurt, Germany, 2000 (`http://www.oonumerics.org/tmpw00/`)

Public Types

- typedef UnaryFunction::result_type **value_type**
- typedef const value_type * **pointer**
- typedef const value_type & **reference**

Public Member Functions

- **processing_iterator** (const Iterator &, const UnaryFunction &)

Dereferencing

- reference **operator** * () const
- pointer **operator** → () const

Random access

- value_type **operator[ ]** (int) const

Increment and decrement

- **processing_iterator** & **operator++** ()
- **processing_iterator operator++** (int)
- **processing_iterator** & **operator–** ()
- **processing_iterator operator–** (int)
- **processing_iterator** & **operator+=** (difference_type)
- **processing_iterator** & **operator-=** (difference_type)
- **processing_iterator operator+** (difference_type)
- **processing_iterator operator-** (difference_type)

Difference

- difference_type **operator-** (const **processing_iterator**< Iterator, UnaryFunction > &)

Comparisons

- bool **operator==** (const **processing_iterator**< Iterator, UnaryFunction > &)
- bool **operator!=** (const **processing_iterator**< Iterator, UnaryFunction > &)
- bool **operator**< (const **processing_iterator**< Iterator, UnaryFunction > &)
- bool **operator**> (const **processing_iterator**< Iterator, UnaryFunction > &)
- bool **operator<=** (const **processing_iterator**< Iterator, UnaryFunction > &)
- bool **operator>=** (const **processing_iterator**< Iterator, UnaryFunction > &)

Public Attributes

- typedef< Iterator >::difference_type **difference_type**

Related Functions

(Note that these are not member functions.)

- **processing_iterator**< Iterator, UnaryFunction > **make_processing_iterator** (Iterator it, UnaryFunction p)

  helper function to create processing iterators

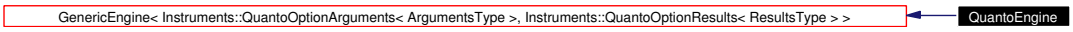
The documentation for this class was generated from the following file:

- ql/Utilities/**processingiterator.hpp**

11.319 QuantoEngine Class Template Reference

`#include <quantoengines.hpp>`

Inheritance diagram for QuantoEngine:



11.319.1 Detailed Description

template<class ArgumentsType, class ResultsType> class QuantLib::PricingEngines::QuantoEngine< ArgumentsType, ResultsType >

Quanto engine base class.

Public Member Functions

- **QuantoEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **calculate** () const

Protected Attributes

- **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

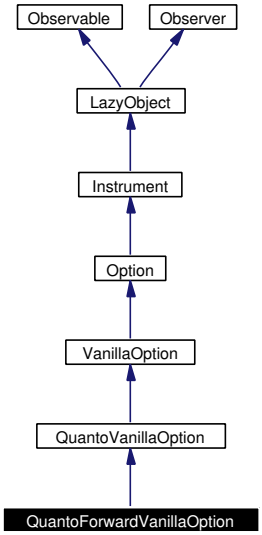
The documentation for this class was generated from the following file:

- ql/PricingEngines/**quantoengines.hpp**

11.320 QuantoForwardVanillaOption Class Reference

`#include <quantoforwardvanillaoption.hpp>`

Inheritance diagram for QuantoForwardVanillaOption:



11.320.1 Detailed Description

Quanto version of a forward vanilla option.

Public Types

- typedef **QuantoOptionArguments**< **ForwardVanillaOption::arguments** > **arguments**
- typedef **QuantoOptionResults**< **ForwardVanillaOption::arguments** > **results**

Public Member Functions

- **QuantoForwardVanillaOption** (Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, double strike, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine, const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateVolTS, const **RelinkableHandle**< **MarketElement** > &correlation, double moneyness, **Date** resetDate, const std::string &isinCode="", const std::string &description="")
- void **setupArguments** (**Arguments** *) const

11.320.2 Member Function Documentation

11.320.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **QuantoVanillaOption** (p. 573).

The documentation for this class was generated from the following files:

- ql/Instruments/**quantoforwardvanillaoption.hpp**
- ql/Instruments/**quantoforwardvanillaoption.cpp**

11.321 QuantoOptionArguments Class Template Reference

`#include <quantovanillaoption.hpp>`

11.321.1 Detailed Description

template<class ArgumentsType> class QuantLib::Instruments::QuantoOptionArguments< ArgumentsType >

arguments for quanto option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- double **correlation**
- **RelinkableHandle< TermStructure > foreignRiskFreeTS**
- **RelinkableHandle< BlackVolTermStructure > exchRateVolTS**

The documentation for this class was generated from the following file:

- ql/Instruments/**quantovanillaoption.hpp**

11.322 QuantoOptionResults Class Template Reference

`#include <quantovanillaoption.hpp>`

11.322.1 Detailed Description

template<class ResultsType> class QuantLib::Instruments::QuantoOptionResults< ResultsType >

results from quanto option calculation

Public Member Functions

- void **reset** ()

Public Attributes

- double **qvega**
- double **qrho**
- double **qlambda**

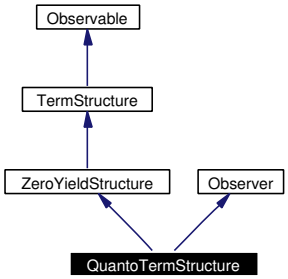
The documentation for this class was generated from the following file:

- ql/Instruments/**quantovanillaoption.hpp**

11.323 QuantoTermStructure Class Reference

`#include <quantotermstructure.hpp>`

Inheritance diagram for QuantoTermStructure:



11.323.1 Detailed Description

Quanto term structure.

Quanto term structure for modelling quanto effect in option pricing.

Note:
This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **QuantoTermStructure** (const **RelinkableHandle**< **TermStructure** > &underlyingDividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &underlyingBlackVolTS, double strike, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateBlackVolTS, double exchRateATMlevel, double underlyingExchRateCorrelation)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date todaysDate** () const
  today's date
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
  the latest date for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const

  returns the zero yield as seen from the evaluation date

11.323.2 Member Function Documentation

11.323.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

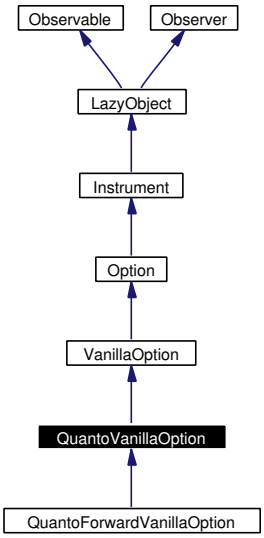
The documentation for this class was generated from the following file:

- ql/TermStructures/**quantotermstructure.hpp**

11.324 QuantoVanillaOption Class Reference

`#include <quantovanillaoption.hpp>`

Inheritance diagram for QuantoVanillaOption:



11.324.1 Detailed Description

quanto version of a vanilla option

Public Types

- typedef **QuantoOptionArguments**< **VanillaOption::arguments** > **arguments**
- typedef **QuantoOptionResults**< **VanillaOption::results** > **results**

Public Member Functions

- **QuantoVanillaOption** (Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, double strike, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine, const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateVolTS, const **RelinkableHandle**< **MarketElement** > &correlation, const std::string &isinCode="", const std::string &description="")
- void **setupArguments** (**Arguments** *) const

greeks

- double **qvega** () const
- double **qrho** () const
- double **qlambda** () const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- **RelinkableHandle**< **TermStructure** > **foreignRiskFreeTS_**
- **RelinkableHandle**< **BlackVolTermStructure** > **exchRateVolTS_**
- **RelinkableHandle**< **MarketElement** > **correlation_**
- double **qvega_**
- double **qrho_**
- double **qlambda_**

11.324.2 Member Function Documentation

11.324.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **VanillaOption** (p. 665).

Reimplemented in **QuantoForwardVanillaOption** (p. 567).

11.324.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from **VanillaOption** (p. 666).

11.324.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **VanillaOption** (p. 666).

The documentation for this class was generated from the following files:

- ql/Instruments/**quantovanillaoption.hpp**
- ql/Instruments/**quantovanillaoption.cpp**

11.325 RandomArrayGenerator Class Template Reference

`#include <randomarraygenerator.hpp>`

11.325.1 Detailed Description

template<class RNG> class QuantLib::RandomNumbers::RandomArrayGenerator< RNG >

Generates random arrays using a random number generator.

Deprecated
use **RandomSequenceGenerator**(p. 575) instead.

Public Types

- typedef **MonteCarlo::Sample**< **Array** > **sample_type**

Public Member Functions

- **RandomArrayGenerator** (const **Array** &variance, long seed=0)
- **RandomArrayGenerator** (const **Math::Matrix** &covariance, long seed=0)
- const **sample_type** & **next** () const
- int **size** () const

The documentation for this class was generated from the following file:

- ql/RandomNumbers/**randomarraygenerator.hpp**

11.326 RandomSequenceGenerator Class Template Reference

`#include <randomsequencegenerator.hpp>`

11.326.1 Detailed Description

template<class RNG> class QuantLib::RandomNumbers::RandomSequenceGenerator< RNG >

Random sequence generator based on a pseudo-random number generator.

Random sequence generator based on a pseudo-random number generator RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Warning:
do not use with low-discrepancy sequence generator

Public Types

- typedef **MonteCarlo::Sample**< **Array** > **sample_type**

Public Member Functions

- **RandomSequenceGenerator** (**Size** dimensionality, const RNG &rng)
- **RandomSequenceGenerator** (**Size** dimensionality, long seed=0)
- const **sample_type** & **nextSequence** () const
- std::vector< unsigned long > **nextInt32Sequence** () const
- const **sample_type** & **lastSequence** () const
- **Size dimension** () const

The documentation for this class was generated from the following file:

- ql/RandomNumbers/**randomsequencegenerator.hpp**

11.327 RateFormatter Class Reference

`#include <dataformatters.hpp>`

11.327.1 Detailed Description

Formats rates for output.

Formatting is in percentage form (xx.xxxxx)

Static Public Member Functions

- std::string **toString** (double rate, int precision=5)

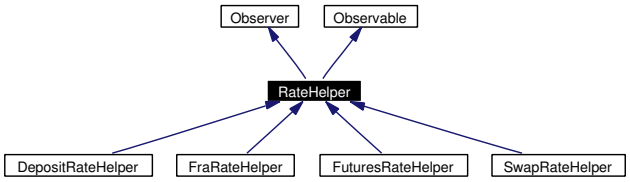
The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.328 RateHelper Class Reference

`#include <ratehelpers.hpp>`

Inheritance diagram for RateHelper:



11.328.1 Detailed Description

base class for rate helpers

This class provides an abstraction for the instruments used to bootstrap a term structure. It is advised that a rate helper for an instrument contains an instance of the actual instrument class to ensure consistancy between the algorithms used during bootstrapping and later instrument pricing. This is not yet fully enforced in the available rate helpers, though - only **SwapRateHelper**(p. 623) contains a Swap instrument for the time being.

Public Member Functions

- **RateHelper** (const **RelinkableHandle**< **MarketElement** > &quote)
- **RateHelper** (double quote)

RateHelper interface

- double **quoteError** () const
- double **referenceQuote** () const
- virtual double **impliedQuote** () const =0
- virtual **DiscountFactor** **discountGuess** () const
- virtual void **setTermStructure** (**TermStructure** *)
  sets the term structure to be used for pricing
- virtual **Date** **maturity** () const =0
  maturity date

Observer interface

- void **update** ()

Protected Attributes

- **RelinkableHandle**< **MarketElement** > **quote_**
- **TermStructure** * **termStructure_**

11.328.2 Member Function Documentation

11.328.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:
Being a pointer and not a **Handle**(p. 390), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented in **DepositRateHelper** (p. 295), **FraRateHelper** (p. 368), and **SwapRateHelper** (p. 624).

11.328.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

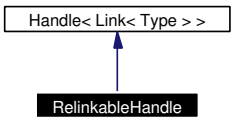
The documentation for this class was generated from the following files:

- ql/TermStructures/**ratehelpers.hpp**
- ql/TermStructures/**ratehelpers.cpp**

11.329 RelinkableHandle Class Template Reference

`#include <relinkablehandle.hpp>`

Inheritance diagram for RelinkableHandle:



11.329.1 Detailed Description

template<class Type> class QuantLib::RelinkableHandle< Type >

Globally accessible relinkable pointer.

An instance of this class can be relinked to another **Handle**(p. 390): such change will be propagated to all the copies of the instance.

Precondition:
Class "Type" must inherit from Observable

Public Member Functions

- **RelinkableHandle** (const **Handle**< Type > &h=**Handle**< Type >(), bool registerAsObserver=true)
- void **linkTo** (const **Handle**< Type > &h, bool registerAsObserver=true)
- const **Handle**< Type > & **operator** → () const

  dereferencing

- bool **isNull** () const

  Checks if the contained handle points to anything.

11.329.2 Constructor & Destructor Documentation

11.329.2.1 RelinkableHandle (const Handle< Type > & *h* = Handle< Type >(), bool *registerAsObserver* = true) `[explicit]`

Warning:
see the documentation of **Link**(p. 455) for issues relatives to `registerAsObserver`.

11.329.3 Member Function Documentation

11.329.3.1 void linkTo (const Handle< Type > & *h*, bool *registerAsObserver* = true)

Warning:
see the documentation of **Link**(p. 455) for issues relatives to `registerAsObserver`.

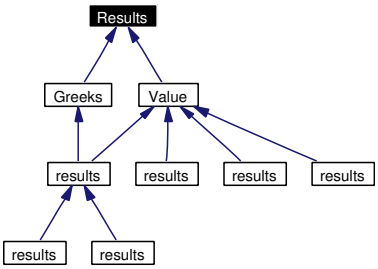
The documentation for this class was generated from the following file:

- ql/**relinkablehandle.hpp**

11.330 Results Class Reference

`#include <argsandresults.hpp>`

Inheritance diagram for Results:



11.330.1 Detailed Description

base class for generic result groups

Public Member Functions

- virtual void **reset** ()=0

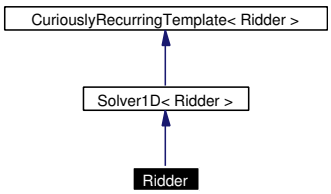
The documentation for this class was generated from the following file:

- ql/**argsandresults.hpp**

11.331 Ridder Class Reference

`#include <ridder.hpp>`

Inheritance diagram for Ridder:



11.331.1 Detailed Description

Ridder 1-D solver

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAcc) const

The documentation for this class was generated from the following file:

- ql/Solvers1D/**ridder.hpp**

11.332 RiskMeasures Class Reference

`#include <riskmeasures.hpp>`

11.332.1 Detailed Description

Deprecated
use Statistics instead

Public Member Functions

- template<class DataIterator> double **percentile** (double percentile, DataIterator begin, DataIterator end) const
- double **gaussianPercentile** (double percentile, double mean, double std) const
- template<class DataIterator> double **potentialUpside** (double percentile, DataIterator begin, DataIterator end) const
- double **gaussianPotentialUpside** (double percentile, double mean, double std) const
- template<class DataIterator> double **valueAtRisk** (double percentile, DataIterator begin, DataIterator end) const
- double **gaussianValueAtRisk** (double percentile, double mean, double std) const
- double **gaussianExpectedShortfall** (double percentile, double mean, double std) const
- double **gaussianShortfall** (double target, double mean, double std) const
- double **gaussianAverageShortfall** (double target, double mean, double std) const

11.332.2 Member Function Documentation

11.332.2.1 double percentile (double *percentile*, DataIterator *begin*, DataIterator *end*) const

Precondition:
percentile must be in range (0%-100%]

11.332.2.2 double gaussianPercentile (double *percentile*, double *mean*, double *std*) const

Precondition:
percentile must be in range (0%-100%) extremes excluded

11.332.2.3 double potentialUpside (double *percentile*, DataIterator *begin*, DataIterator *end*) const

Precondition:
y must be in range [90%-100%)

11.332.2.4 double gaussianPotentialUpside (double *percentile*, double *mean*, double *std*) const

Precondition:
percentile must be in range [90%-100%)

11.332.2.5 double valueAtRisk (double *percentile*, DataIterator *begin*, DataIterator *end*) const

Precondition:
y must be in range [90%-100%)

11.332.2.6 double gaussianValueAtRisk (double *percentile*, double *mean*, double *std*) const

Precondition:
percentile must be in range [90%-100%)

11.332.2.7 double gaussianExpectedShortfall (double *percentile*, double *mean*, double *std*) const

Precondition:
percentile must be in range 90%-100%

The documentation for this class was generated from the following file:

- ql/Math/**riskmeasures.hpp**

11.333 RiskStatistics_old Class Reference

`#include <riskstatistics.hpp>`

11.333.1 Detailed Description

Risk analysis tool.

It can accumulate a set of data and return statistics as mean, variance, std. deviation, skewness, kurtosis, plus risk quantities as Value-At-Risk, Expected Shortfall, Shortfall, Average Shortfall, etc under gaussian assumption

Deprecated
use IncrementalGaussianStatistics or Statistics instead

Public Member Functions

Inspectors

- **Size samples** () const
- double **weightSum** () const
- double **mean** () const
- double **variance** () const
- double **standardDeviation** () const
- double **errorEstimate** () const
- double **skewness** () const
- double **kurtosis** () const
- double **min** () const
- double **max** () const
- double **potentialUpside** (double percentile) const
  returns the Potential-Upside at a given percentile
- double **valueAtRisk** (double percentile) const
  returns the Value-At-Risk at a given percentile
- double **expectedShortfall** (double percentile) const
  returns the Expected Shortfall at a given percentile
- double **shortfall** (double target) const
  returns the Shortfall (observations below target)
- double **averageShortfall** (double target) const
  returns the Average Shortfall (averaged shortfallness)

Modifiers

- void **add** (double value, double weight=1.0)
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
  adds a sequence of data to the set, each with its weight
- void **reset** ()

11.333.2 Member Function Documentation

11.333.2.1 void add (double *value*, double *weight* = 1.0)

Precondition:
weights must be positive or null

The documentation for this class was generated from the following file:

- ql/riskstatistics.hpp

11.334 Sample Struct Template Reference

`#include <sample.hpp>`

11.334.1 Detailed Description

template<class T> struct QuantLib::MonteCarlo::Sample< T >

weighted sample

Public Types

- typedef T **value_type**

Public Member Functions

- **Sample** (const T &value, double weight)

Public Attributes

- T **value**
- double **weight**

The documentation for this struct was generated from the following file:

- ql/MonteCarlo/**sample.hpp**

11.335 Schedule Class Reference

`#include <scheduler.hpp>`

11.335.1 Detailed Description

Payment schedule.

Iterators

- typedef std::vector< **Date** >::const_iterator **const_iterator**
- const_iterator **begin** () const
- const_iterator **end** () const

Public Member Functions

- **Schedule** (const **Calendar** &calendar, const **Date** &startDate, const **Date** &endDate, int frequency, **RollingConvention** rollingConvention, bool isAdjusted, const **Date** &stubDate=**Date**(), bool startFromEnd=false, bool longFinal=false)
- **Schedule** (const std::vector< **Date** > &, const **Calendar** &calendar, **RollingConvention** rollingConvention, bool isAdjusted)

Date access

- **Size size** () const
- const **Date** & **operator[ ]** (**Size** i) const
- const **Date** & **date** (**Size** i) const
- bool **isRegular** (**Size** i) const

Other inspectors

- const **Calendar** & **calendar** () const
- const **Date** & **startDate** () const
- const **Date** & **endDate** () const
- int **frequency** () const
- **RollingConvention rollingConvention** () const
- bool **isAdjusted** () const

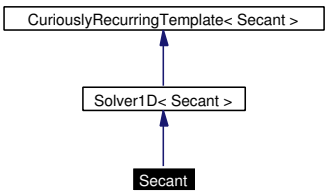
The documentation for this class was generated from the following files:

- ql/**scheduler.hpp**
- ql/**scheduler.cpp**

11.336 Secant Class Reference

`#include <secant.hpp>`

Inheritance diagram for Secant:



11.336.1 Detailed Description

secant 1-D solver

Public Member Functions

- template<class F> double **solveImpl** (const F &f, double xAccuracy) const

The documentation for this class was generated from the following file:

- ql/Solvers1D/**secant.hpp**

11.337 SegmentIntegral Class Reference

`#include <segmentintegral.hpp>`

11.337.1 Detailed Description

Integral of a one-dimensional function.

Given a number N of intervals, the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f \mathrm{d}x = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b - a)/N$.

Examples:
EuropeanOption.cpp.

Public Member Functions

- **SegmentIntegral** (**Size** intervals)
- template<class F> double **operator()** (const F &f, double a, double b) const

The documentation for this class was generated from the following file:

- ql/Math/**segmentintegral.hpp**

11.338 SequenceStatistics Class Template Reference

`#include <sequencestatistics.hpp>`

11.338.1 Detailed Description

template<class StatisticsType = Statistics> class QuantLib::Math::SequenceStatistics< StatisticsType >

Statistics analysis of N-dimensional (sequence) data.

It provides 1-dimensional statistics as discrepancy plus N-dimensional (sequence) statistics (e.g. mean, variance, skewness, kurtosis, etc.) with one component for each dimension of the sample space.

For most of the statistics this class relies on the StatisticsType underlying class to provide 1-D methods that will be iterated for all the components of the N-D data. These lifted methods are the union of all the methods that might be requested to the 1-D underlying StatisticsType class, with the usual compile-time checks provided by the template approach.

Public Types

- typedef StatisticsType **statistics_type**

Public Member Functions

- **SequenceStatistics** (**Size** dimension)

covariance and correlation

- **Disposable**< **Matrix** > **covariance** () const
  returns the covariance **Matrix**(p. 471)
- **Disposable**< **Matrix** > **correlation** () const
  returns the correlation **Matrix**(p. 471)

1-D inspectors lifted from underlying statistics class

- **Size samples** () const
- double **weightSum** () const

N-D inspectors lifted from underlying statistics class

- std::vector< double > **mean** () const
- std::vector< double > **variance** () const
- std::vector< double > **standardDeviation** () const
- std::vector< double > **downsideVariance** () const
- std::vector< double > **downsideDeviation** () const
- std::vector< double > **semiVariance** () const
- std::vector< double > **semiDeviation** () const
- std::vector< double > **errorEstimate** () const

- std::vector< double > **skewness** () const
- std::vector< double > **kurtosis** () const
- std::vector< double > **min** () const
- std::vector< double > **max** () const
- std::vector< double > **gaussianPercentile** (double y) const
- std::vector< double > **percentile** (double y) const
- std::vector< double > **gaussianPotentialUpside** (double percentile) const
- std::vector< double > **potentialUpside** (double percentile) const
- std::vector< double > **gaussianValueAtRisk** (double percentile) const
- std::vector< double > **valueAtRisk** (double percentile) const
- std::vector< double > **gaussianExpectedShortfall** (double percentile) const
- std::vector< double > **expectedShortfall** (double percentile) const
- std::vector< double > **regret** (double target) const
- std::vector< double > **gaussianShortfall** (double target) const
- std::vector< double > **shortfall** (double target) const
- std::vector< double > **gaussianAverageShortfall** (double target) const
- std::vector< double > **averageShortfall** (double target) const

Modifiers

- void **reset** (**Size** dimension=0)
- template<class Sequence> void **add** (const Sequence &sample, double weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, double weight=1.0)

Protected Attributes

- **Size dimension_**
- std::vector< statistics_type > **stats_**
- std::vector< double > **results_**
- **Matrix quadraticSum_**

The documentation for this class was generated from the following file:

- ql/Math/**sequencestatistics.hpp**

11.339 Short Class Template Reference

`#include <shortindexedcoupon.hpp>`

11.339.1 Detailed Description

template<class IndexedCouponType> class QuantLib::CashFlows::Short< IndexedCouponType >

short indexed coupon

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **Short** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())
- double **amount** () const

  inhibit calculation

11.339.2 Member Function Documentation

11.339.2.1 double amount () const

inhibit calculation

Unlike **ParCoupon**(p. 541), this coupon can't calculate its fixing for future dates, either.

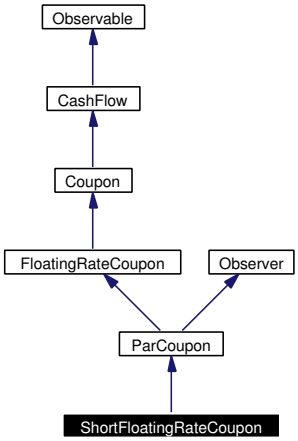
The documentation for this class was generated from the following file:

- ql/CashFlows/**shortindexedcoupon.hpp**

11.340 ShortFloatingRateCoupon Class Reference

`#include <shortfloatingcoupon.hpp>`

Inheritance diagram for ShortFloatingRateCoupon:



11.340.1 Detailed Description

short coupon at par on a term structure

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ShortFloatingRateCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())
- double **amount** () const

  throws when an interpolated fixing is needed

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

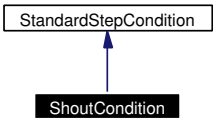
The documentation for this class was generated from the following files:

- ql/CashFlows/**shortfloatingcoupon.hpp**
- ql/CashFlows/**shortfloatingcoupon.cpp**

11.341 ShoutCondition Class Reference

`#include <shoutcondition.hpp>`

Inheritance diagram for ShoutCondition:



11.341.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

Todo

Unify the intrinsicValues/Payoff thing

Public Member Functions

- **ShoutCondition** (Option::Type type, double strike, **Time** resTime, **Rate** rate)
- **ShoutCondition** (const **Array** &intrinsicValues, **Time** resTime, **Rate** rate)
- void **applyTo** (**Array** &a, **Time** t) const
- void **applyTo** (**Handle**< **DiscretizedAsset** > asset) const

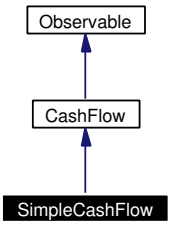
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**shoutcondition.hpp**

11.342 SimpleCashFlow Class Reference

`#include <simplecashflow.hpp>`

Inheritance diagram for SimpleCashFlow:



11.342.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **SimpleCashFlow** (double amount, const **Date** &date)

CashFlow interface

- double **amount** () const
  returns the amount of the cash flow
- **Date date** () const
  returns the date at which the cash flow is settled

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

11.342.2 Member Function Documentation

11.342.2.1 double amount () const [virtual]

returns the amount of the cash flow

Note:
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. 252).

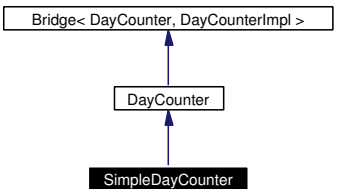
The documentation for this class was generated from the following file:

- ql/CashFlows/**simplecashflow.hpp**

11.343 SimpleDayCounter Class Reference

`#include <simpledaycounter.hpp>`

Inheritance diagram for SimpleDayCounter:



11.343.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

This day counter tries to ensure that whole-month distances are returned as a simple fraction, i.e., 1 year = 1.0, 6 months = 0.5, 3 months = 0.25 and so forth.

Warning:
this day counter should be used together with NullCalendar, which ensures that dates at whole-month distances share the same day of month. It is **not** guaranteed to work with any other calendar.

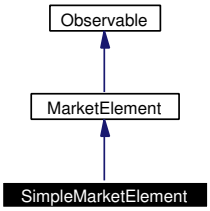
The documentation for this class was generated from the following file:

- ql/DayCounters/**simpledaycounter.hpp**

11.344 SimpleMarketElement Class Reference

`#include <marketelement.hpp>`

Inheritance diagram for SimpleMarketElement:



11.344.1 Detailed Description

market element returning a stored value

Public Member Functions

- **SimpleMarketElement** (double value)

Market element interface

- double **value** () const
  returns the current value

Modifiers

- void **setValue** (double value)

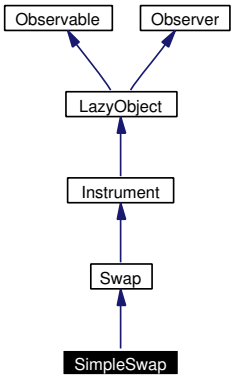
The documentation for this class was generated from the following file:

- ql/**marketelement.hpp**

11.345 SimpleSwap Class Reference

`#include <simpleswap.hpp>`

Inheritance diagram for SimpleSwap:



11.345.1 Detailed Description

Simple fixed-rate vs Libor swap.

Public Member Functions

- **SimpleSwap** (bool payFixedRate, const **Date** &startDate, int n, **TimeUnit** units, const **Calendar** &calendar, **RollingConvention** rollingConvention, double nominal, int fixedFrequency, **Rate** fixedRate, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency, const **Handle**< **Indexes::Xibor** > &index, int indexFixingDays, **Spread** spread, const **RelinkableHandle**< **TermStructure** > &termStructure, const std::string &isinCode="", const std::string &description="")
- **SimpleSwap** (bool payFixedRate, double nominal, const **Schedule** &fixedSchedule, **Rate** fixedRate, const **DayCounter** &fixedDayCount, const **Schedule** &floatSchedule, const **Handle**< **Indexes::Xibor** > &index, int indexFixingDays, **Spread** spread, const **RelinkableHandle**< **TermStructure** > &termStructure, const std::string &isinCode="", const std::string &description="")
- **SimpleSwap** (bool payFixedRate, const **Date** &startDate, const **Date** &maturity, const **Calendar** &calendar, **RollingConvention** rollingConvention, double nominal, int fixedFrequency, **Rate** fixedRate, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency, const **Handle**< **Indexes::Xibor** > &index, int indexFixingDays, **Spread** spread, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Date** &fixedStubDate=**Date**(), bool fixedFromEnd=false, bool fixedLongFinal=false, const **Date** &floatStubDate=**Date**(), bool floatFromEnd=false, bool floatLongFinal=false, const std::string &isinCode="", const std::string &description="")
- **Rate fairRate** () const
- **Spread fairSpread** () const
- double **fixedLegBPS** () const
- double **floatingLegBPS** () const
- **Rate fixedRate** () const
- **Spread spread** () const

- double **nominal** () const
- bool **payFixedRate** () const
- const std::vector< **Handle**< **CashFlow** > > & **fixedLeg** () const
- const std::vector< **Handle**< **CashFlow** > > & **floatingLeg** () const
- void **setupArguments** (**Arguments** *args) const

11.345.2 Constructor & Destructor Documentation

11.345.2.1 SimpleSwap (bool *payFixedRate*, const Date & *startDate*, const Date & *maturity*, const Calendar & *calendar*, RollingConvention *rollingConvention*, double *nominal*, int *fixedFrequency*, Rate *fixedRate*, bool *fixedIsAdjusted*, const DayCounter & *fixedDayCount*, int *floatingFrequency*, const Handle< Indexes::Xibor > & *index*, int *indexFixingDays*, Spread *spread*, const RelinkableHandle< TermStructure > & *termStructure*, const Date & *fixedStubDate* = Date(), bool *fixedFromEnd* = false, bool *fixedLongFinal* = false, const Date & *floatStubDate* = Date(), bool *floatFromEnd* = false, bool *floatLongFinal* = false, const std::string & *isinCode* = "", const std::string & *description* = "")

Deprecated
use the constructor taking two Schedules

11.345.3 Member Function Documentation

11.345.3.1 void setupArguments (Arguments * *args*) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

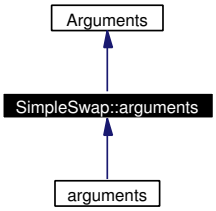
The documentation for this class was generated from the following files:

- ql/Instruments/**simpleswap.hpp**
- ql/Instruments/**simpleswap.cpp**

11.346 SimpleSwap::arguments Class Reference

`#include <simpleswap.hpp>`

Inheritance diagram for SimpleSwap::arguments:



11.346.1 Detailed Description

arguments for simple swap calculation

Public Member Functions

- void **validate** () const

Public Attributes

- bool **payFixed**
- double **nominal**
- std::vector< **Time** > **fixedResetTimes**
- std::vector< **Time** > **fixedPayTimes**
- std::vector< double > **fixedCoupons**
- std::vector< **Time** > **floatingAccrualTimes**
- std::vector< **Time** > **floatingResetTimes**
- std::vector< **Time** > **floatingPayTimes**
- std::vector< **Spread** > **floatingSpreads**

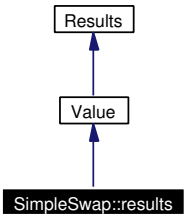
The documentation for this class was generated from the following files:

- ql/Instruments/**simpleswap.hpp**
- ql/Instruments/**simpleswap.cpp**

11.347 SimpleSwap::results Class Reference

`#include <simpleswap.hpp>`

Inheritance diagram for SimpleSwap::results:



11.347.1 Detailed Description

results from swaption calculation

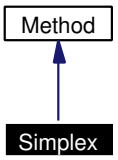
The documentation for this class was generated from the following file:

- ql/Instruments/**simpleswap.hpp**

11.348 Simplex Class Reference

`#include <simplex.hpp>`

Inheritance diagram for Simplex:



11.348.1 Detailed Description

Multi-dimensional **Simplex**(p. 603) class.

Public Member Functions

- **Simplex** (double lambda, double tol)
- virtual void **minimize** (const **Problem** &P) const

  minimize the optimization problem P

11.348.2 Constructor & Destructor Documentation

11.348.2.1 Simplex (double *lambda*, double *tol*)

Constructor taking as input lambda as the characteristic length and tol as the precision

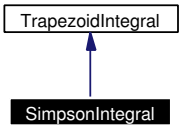
The documentation for this class was generated from the following files:

- ql/Optimization/**simplex.hpp**
- ql/Optimization/**simplex.cpp**

11.349 SimpsonIntegral Class Reference

`#include <simpsonintegral.hpp>`

Inheritance diagram for SimpsonIntegral:



11.349.1 Detailed Description

Integral of a one-dimensional function.

Public Member Functions

- **SimpsonIntegral** (double accuracy, **Size** maxIterations=**Null**< int >())
- template<class F> double **operator()** (const F &f, double a, double b) const

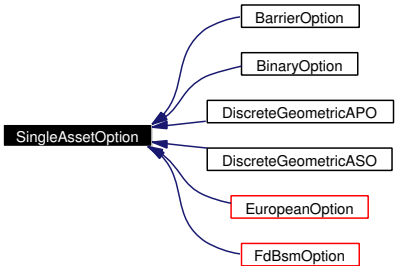
The documentation for this class was generated from the following file:

- ql/Math/simpsonintegral.hpp

11.350 SingleAssetOption Class Reference

`#include <singleassetoption.hpp>`

Inheritance diagram for SingleAssetOption:



11.350.1 Detailed Description

Black-Scholes-Merton option.

Public Member Functions

- **SingleAssetOption** (Option::Type type, double underlying, double strike, **Spread** dividendYield, **Rate** riskFreeRate, **Time** residualTime, double volatility)
- virtual void **setVolatility** (double newVolatility)
- virtual void **setRiskFreeRate** (**Rate** newRate)
- virtual void **setDividendYield** (**Rate** newDividendYield)
- virtual double **value** () const =0
- virtual double **delta** () const =0
- virtual double **gamma** () const =0
- virtual double **theta** () const
- virtual double **vega** () const
- virtual double **rho** () const
- virtual double **dividendRho** () const
- double **impliedVolatility** (double targetValue, double accuracy=1e-4, **Size** maxEvaluations=100, double minVol=QL_MIN_VOLATILITY, double maxVol=QL_MAX_VOLATILITY) const
- double **impliedDivYield** (double targetValue, double accuracy=1e-4, **Size** maxEvaluations=100, double minVol=QL_MIN_DIVYIELD, double maxVol=QL_MAX_DIVYIELD) const
- virtual **Handle**< **SingleAssetOption** > **clone** () const =0

Protected Attributes

- double **underlying_**
- **PlainVanillaPayoff payoff_**
- **Spread dividendYield_**
- **Rate riskFreeRate_**
- **Time residualTime_**

- double **volatility_**
- bool **hasBeenCalculated_**
- double **rho_**
- double **dividendRho_**
- double **vega_**
- double **theta_**
- bool **rhoComputed_**
- bool **dividendRhoComputed_**
- bool **vegaComputed_**
- bool **thetaComputed_**

Static Protected Attributes

- const double **dVolMultiplier_** = 0.0001
- const double **dRMultiplier_** = 0.0001

Friends

- class **VolatilityFunction**
- class **DivYieldFunction**

11.350.2 Member Function Documentation

11.350.2.1 double impliedVolatility (double *targetValue*, double *accuracy* = 1e-4, Size *maxEvaluations* = 100, double *minVol* = QL_MIN_VOLATILITY, double *maxVol* = QL_MAX_VOLATILITY) const

Warning:

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases impliedVolatility can fail and in any case is meaningless. Another possible source of failure is to have a targetValue that is not attainable with any volatility, e.g. a targetValue lower than the intrinsic value in the case of American options.

The documentation for this class was generated from the following files:

- ql/Pricers/**singleassetoption.hpp**
- ql/Pricers/**singleassetoption.cpp**

11.351 SobolRsg Class Reference

`#include <sobolrsg.hpp>`

11.351.1 Detailed Description

Sobol low-discrepancy sequence generator.

A Gray code counter and bitwise operations are used for very fast sequence generation.

The implementation relies on primitive polynomials modulo two and initialization numbers from the book "Monte Carlo Methods in Finance" by Peter Jäckel.

21200 primitive polynomials modulo two are provided by default. There are 8 129 334 polynomials as provided by Jäckel which can be downloaded from quantlib.org. If you need that many dimensions you must replace the primitivepolynomial.* files with the ones downloaded and recompile the library.

The choice of initialization numbers is crucial for the homogeneity properties of the sequence. Jäckel's initialization numbers are superior to the "unit initialization" suggested in "Numerical Recipes in C" by Press, Teukolsky, Vetterling, and Flannery.

For more info on Sobol sequences see "Monte Carlo Methods in Finance", by Peter Jäckel, section 8.3 and "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery, section 7.7.

Public Types

- typedef **MonteCarlo::Sample**< **Array** > **sample_type**

Public Member Functions

- **SobolRsg** (**Size** dimensionality, unsigned long seed=0, bool unitInitialization=false)
- const **sample_type** & **nextSequence** () const
- const **sample_type** & **lastSequence** () const
- **Size dimension** () const

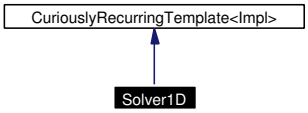
The documentation for this class was generated from the following files:

- ql/RandomNumbers/**sobolrsg.hpp**
- ql/RandomNumbers/**sobolrsg.cpp**

11.352 Solver1D Class Template Reference

`#include <solver1d.hpp>`

Inheritance diagram for Solver1D:



11.352.1 Detailed Description

template<class Impl> class QuantLib::Solver1D< Impl >

Base class for 1-D solvers.

The implementation of this class uses the so-called "Barton-Nackman trick", also known as "the curiously recurring template pattern". Concrete solvers will be declared as:

```
class Foo : public Solver1D<Foo> {
  public:
    ...
    template <class F>
    double solveImpl(const F& f, double accuracy) const {
        ...
    }
};
```

Before calling `solveImpl`, the base class will set its protected data members so that:

- `xMin_` and `xMax_` form a valid bracket;
- `fxMin_` and `fxMax_` contain the values of the function in `xMin_` and `xMax_`;
- `root_` is a valid initial guess. The implementation of `solveImpl` can safely assume all of the above.

Public Member Functions

Modifiers

- template<class F> double **solve** (const F &f, double accuracy, double guess, double step) const
- template<class F> double **solve** (const F &f, double accuracy, double guess, double xMin, double xMax) const
- void **setMaxEvaluations** (**Size** evaluations)
- void **setLowerBound** (double lowerBound)

  sets the lower bound for the function domain

- void **setUpperBound** (double upperBound)

  sets the upper bound for the function domain

Protected Attributes

- double **root_**
- double **xMin_**
- double **xMax_**
- double **fxMin_**
- double **fxMax_**
- **Size maxEvaluations_**
- **Size evaluationNumber_**

11.352.2 Member Function Documentation

11.352.2.1 double solve (const F & *f*, double *accuracy*, double *guess*, double *step*) const

This method returns the zero of the function f, determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

11.352.2.2 double solve (const F & *f*, double *accuracy*, double *guess*, double *xMin*, double *xMax*) const

This method returns the zero of the function f, determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). An initial guess must be supplied, as well as two values $x_{\min}$ and $x_{\max}$ which must bracket the zero (i.e., either $f(x_{\min}) \leq 0 \leq f(x_{\max})$, or $f(x_{\max}) \leq 0 \leq f(x_{\min})$ must be true).

11.352.2.3 void setMaxEvaluations (Size *evaluations*)

This method sets the maximum number of function evaluations for the bracketing routine. An error is thrown if a bracket is not found after this number of evaluations.

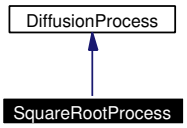
The documentation for this class was generated from the following file:

- ql/**solver1d.hpp**

11.353 SquareRootProcess Class Reference

`#include <diffusionprocess.hpp>`

Inheritance diagram for SquareRootProcess:



11.353.1 Detailed Description

Square-root process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma \sqrt{x_t} dW_t.$$

Public Member Functions

- **SquareRootProcess** (double b, double a, double sigma, double x0=0)
- double **drift** (**Time** t, double x) const

  returns the drift part of the equation, i.e. $\mu(t, x_t)$

- double **diffusion** (**Time** t, double x) const

  returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

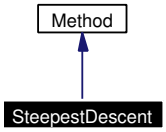
The documentation for this class was generated from the following file:

- ql/**diffusionprocess.hpp**

11.354 SteepestDescent Class Reference

`#include <steepestdescent.hpp>`

Inheritance diagram for SteepestDescent:



11.354.1 Detailed Description

Multi-dimensional steepest-descent class.

User has to provide line-search method and optimization end criteria

search direction = $-f'(x)$

Public Member Functions

- **SteepestDescent** ()
  default default constructor (msvc bug)
- **SteepestDescent** (const **Handle**< **LineSearch** > &lineSearch)
  default constructor
- virtual ~**SteepestDescent** ()
  destructor
- virtual void **minimize** (const **Problem** &P) const
  minimize the optimization problem P

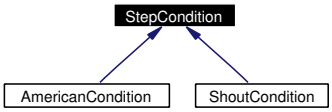
The documentation for this class was generated from the following files:

- ql/Optimization/**steepestdescent.hpp**
- ql/Optimization/**steepestdescent.cpp**

11.355 StepCondition Class Template Reference

`#include <stepcondition.hpp>`

Inheritance diagram for StepCondition:



11.355.1 Detailed Description

template<class arrayType> class QuantLib::FiniteDifferences::StepCondition< arrayType >

condition to be applied at every time step

Public Member Functions

- virtual void **applyTo** (arrayType &a, **Time** t) const =0
- virtual void **applyTo** (**Handle**< **DiscretizedAsset** > asset) const =0

The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**stepcondition.hpp**

11.356 stepping_iterator Class Template Reference

`#include <steppingiterator.hpp>`

11.356.1 Detailed Description

template<class RandomAccessIterator> class QuantLib::Utilities::stepping_iterator< RandomAccessIterator >

Iterator advancing in constant steps.

This iterator advances an underlying random access iterator in steps of n positions, where n is an integer given upon construction.

Public Member Functions

- **stepping_iterator** (const RandomAccessIterator &it, difference_type step)

Dereferencing

- reference **operator** * () const
- pointer **operator** → () const

Random access

- reference **operator[ ]** (int i) const

Increment and decrement

- **stepping_iterator** & **operator++** ()
- **stepping_iterator operator++** (int)
- **stepping_iterator** & **operator–** ()
- **stepping_iterator operator–** (int)
- **stepping_iterator** & **operator+=** (difference_type i)
- **stepping_iterator** & **operator-=** (difference_type i)
- **stepping_iterator operator+** (difference_type i)
- **stepping_iterator operator-** (difference_type i)

Difference

- difference_type **operator-** (const **stepping_iterator** &i)

Comparisons

- bool **operator==** (const **stepping_iterator** &i)
- bool **operator!=** (const **stepping_iterator** &i)
- bool **operator**< (const **stepping_iterator** &i)
- bool **operator**> (const **stepping_iterator** &i)
- bool **operator**<= (const **stepping_iterator** &i)
- bool **operator**>= (const **stepping_iterator** &i)

Public Attributes

- typedef< RandomAccessIterator >::difference_type **difference_type**
- typedef< RandomAccessIterator >::pointer **pointer**
- typedef< RandomAccessIterator >::reference **reference**

Related Functions

(Note that these are not member functions.)

- **stepping_iterator**< Iterator > **make_stepping_iterator** (Iterator it, typename **stepping_iterator**< Iterator >::difference_type step)

  helper function to create stepping iterators

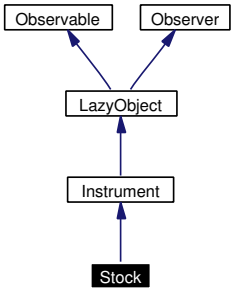
The documentation for this class was generated from the following file:

- ql/Utilities/**steppingiterator.hpp**

11.357 Stock Class Reference

`#include <stock.hpp>`

Inheritance diagram for Stock:



11.357.1 Detailed Description

Simple stock class.

Public Member Functions

- **Stock** (const **RelinkableHandle**< **MarketElement** > &quote, const std::string &isinCode, const std::string &description)
- bool **isExpired** () const

  returns whether the instrument is still tradable.

Protected Member Functions

- void **performCalculations** () const

11.357.2 Member Function Documentation

11.357.2.1 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. 424).

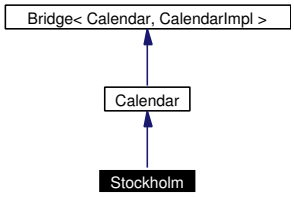
The documentation for this class was generated from the following files:

- ql/Instruments/**stock.hpp**
- ql/Instruments/**stock.cpp**

11.358 Stockholm Class Reference

`#include <stockholm.hpp>`

Inheritance diagram for Stockholm:



11.358.1 Detailed Description

Stockholm calendar

Holidays:

- Saturdays
- Sundays
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- Midsummer Eve (Friday between June 18-24)
- New Year's Day, January 1st
- Epiphany, January 6th
- May Day, May 1st
- National Day, June 6th
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

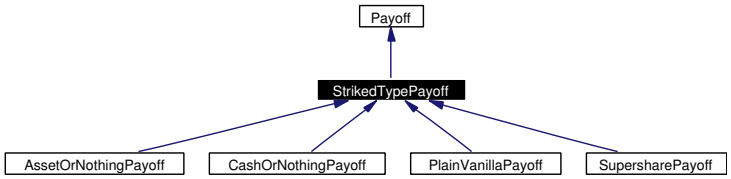
The documentation for this class was generated from the following file:

- ql/Calendars/**stockholm.hpp**

11.359 StrikedTypePayoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for StrikedTypePayoff:



11.359.1 Detailed Description

We need an intermediate class for all those payoff that have both a strike and a type but are not necessarily plain-vanilla

Public Member Functions

- **StrikedTypePayoff** (Option::Type type, double strike)
- Option::Type **optionType** () const
- double **strike** () const

Protected Attributes

- Option::Type **type_**
- double **strike_**

The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.360 StringFormatter Class Reference

`#include <dataformatters.hpp>`

11.360.1 Detailed Description

Formats strings as lower- or uppercase.

Static Public Member Functions

- std::string **toLowercase** (const std::string &s)
- std::string **toUppercase** (const std::string &s)

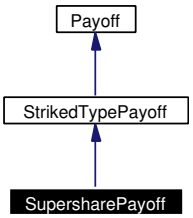
The documentation for this class was generated from the following files:

- ql/**dataformatters.hpp**
- ql/**dataformatters.cpp**

11.361 SupersharePayoff Class Reference

`#include <payoff.hpp>`

Inheritance diagram for SupersharePayoff:



11.361.1 Detailed Description

Binary(p. 195) supershare option payoff.

Public Member Functions

- **SupersharePayoff** (Option::Type type, double strike, double strikeIncrement)
- double **operator()** (double price) const

The documentation for this class was generated from the following file:

- ql/**payoff.hpp**

11.362 SVD Class Reference

`#include <svd.hpp>`

11.362.1 Detailed Description

Singular **Value**(p. 662) Decomposition.

Refer to Golub and Van Loan: **Matrix**(p. 471) computation, The Johns Hopkins University Press

Public Member Functions

- **SVD** (const **Matrix** &Arg)
- void **getU** (**Matrix** &A) const
- void **getV** (**Matrix** &A) const
- void **getSingularValues** (**Array** &x) const
- void **getS** (**Matrix** &S) const
- double **norm2** ()
- double **cond** ()
- int **rank** ()

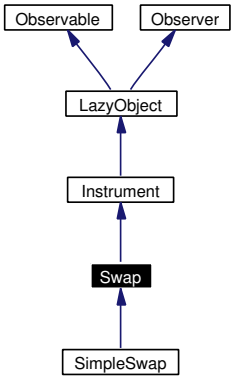
The documentation for this class was generated from the following files:

- ql/Math/**svd.hpp**
- ql/Math/**svd.cpp**

11.363 Swap Class Reference

`#include <swap.hpp>`

Inheritance diagram for Swap:



11.363.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are payed; the ones belonging to the first leg are received.

Public Member Functions

- **Swap** (const std::vector< **Handle**< **CashFlow** > > &firstLeg, const std::vector< **Handle**< **CashFlow** > > &secondLeg, const **RelinkableHandle**< **TermStructure** > &termStructure, const std::string &isinCode="", const std::string &description="")

Instrument interface

- bool **isExpired** () const
  returns whether the instrument is still tradable.

Additional interface

- **Date startDate** () const
- **Date maturity** () const
- double **firstLegBPS** () const
- double **secondLegBPS** () const
- **CashFlows::TimeBasket sensitivity** (int basis=2) const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- std::vector< **Handle**< **CashFlow** > > **firstLeg_**
- std::vector< **Handle**< **CashFlow** > > **secondLeg_**
- **RelinkableHandle**< **TermStructure** > **termStructure_**
- double **firstLegBPS_**
- double **secondLegBPS_**

11.363.2 Member Function Documentation

11.363.2.1 TimeBasket sensitivity (int *basis* = 2) const

Unstable
This method was not tested enough to guarantee its functionality.

11.363.2.2 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from **Instrument** (p. 424).

11.363.2.3 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. 424).

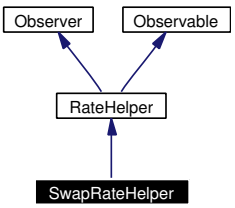
The documentation for this class was generated from the following files:

- ql/Instruments/**swap.hpp**
- ql/Instruments/**swap.cpp**

11.364 SwapRateHelper Class Reference

`#include <ratehelpers.hpp>`

Inheritance diagram for SwapRateHelper:



11.364.1 Detailed Description

Swap rate.

Warning:
This class assumes that the settlement date does not change between calls of **setTermStructure()**(p. 624).

Public Member Functions

- **SwapRateHelper** (const **RelinkableHandle**< **MarketElement** > &rate, int n, **TimeUnit** units, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, int fixedFrequency, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency)
- **SwapRateHelper** (double rate, int n, **TimeUnit** units, int settlementDays, const **Calendar** &calendar, **RollingConvention** convention, int fixedFrequency, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency)
- double **impliedQuote** () const
- **Date maturity** () const

  maturity date

- void **setTermStructure** (**TermStructure** *)

  sets the term structure to be used for pricing

Protected Attributes

- int **n_**
- **TimeUnit units_**
- int **settlementDays_**
- **Calendar calendar_**
- **RollingConvention convention_**
- int **fixedFrequency_**
- int **floatingFrequency_**
- bool **fixedIsAdjusted_**

- **DayCounter fixedDayCount_**
- **Date settlement_**
- **Handle< Instruments::SimpleSwap > swap_**
- **RelinkableHandle< TermStructure > termStructureHandle_**

11.364.2 Member Function Documentation

11.364.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:
Being a pointer and not a **Handle**(p. 390), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. 578).

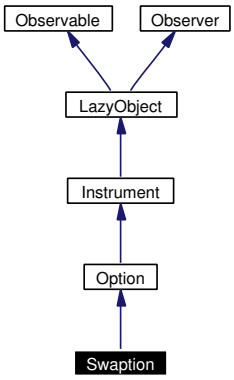
The documentation for this class was generated from the following files:

- ql/TermStructures/**ratehelpers.hpp**
- ql/TermStructures/**ratehelpers.cpp**

11.365 Swaption Class Reference

`#include <swaption.hpp>`

Inheritance diagram for Swaption:



11.365.1 Detailed Description

Swaption(p. 625) class.

Public Member Functions

- **Swaption** (const **Handle**< **SimpleSwap** > &swap, const **Exercise** &exercise, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)
- bool **isExpired** () const

  returns whether the instrument is still tradable.

- void **setupArguments** (**Arguments** *) const

Protected Member Functions

- void **performCalculations** () const

11.365.2 Member Function Documentation

11.365.2.1 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

11.365.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. 424).

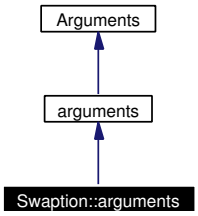
The documentation for this class was generated from the following files:

- ql/Instruments/**swaption.hpp**
- ql/Instruments/**swaption.cpp**

11.366 Swaption::arguments Class Reference

`#include <swaption.hpp>`

Inheritance diagram for Swaption::arguments:



11.366.1 Detailed Description

arguments for swaption calculation

Public Member Functions

- void **validate** () const

Public Attributes

- **Rate fairRate**
- **Rate fixedRate**
- double **fixedBPS**
- Exercise::Type **exerciseType**
- std::vector< **Time** > **exerciseTimes**

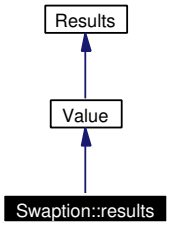
The documentation for this class was generated from the following files:

- ql/Instruments/**swaption.hpp**
- ql/Instruments/**swaption.cpp**

11.367 Swaption::results Class Reference

`#include <swaption.hpp>`

Inheritance diagram for Swaption::results:



11.367.1 Detailed Description

results from swaption calculation

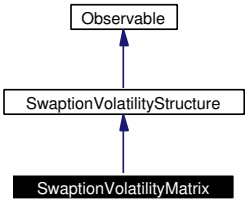
The documentation for this class was generated from the following file:

- ql/Instruments/**swaption.hpp**

11.368 SwaptionVolatilityMatrix Class Reference

`#include <swaptionvolmatrix.hpp>`

Inheritance diagram for SwaptionVolatilityMatrix:



11.368.1 Detailed Description

Swaption at-the-money volatility matrix.

This class provides the at-the-money volatility for a given swaption by interpolating a volatility matrix whose elements are the market volatilities of a set of swaption with given exercise date and length.

Todo

Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Public Member Functions

- **SwaptionVolatilityMatrix** (const **Date** &todaysDate, const std::vector< **Date** > &exerciseDates, const std::vector< **Period** > &lengths, const **Math::Matrix** &volatilities, const **DayCounter** &dayCounter=**DayCounters::Thirty360**())
- **Date todaysDate** () const

  returns today's date

- **DayCounter dayCounter** () const

  returns the day counter used for internal date/time conversions

- const std::vector< **Date** > & **exerciseDates** () const
- const std::vector< **Period** > & **lengths** () const

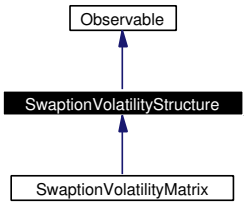
The documentation for this class was generated from the following file:

- ql/Volatilities/**swaptionvolmatrix.hpp**

11.369 SwaptionVolatilityStructure Class Reference

`#include <swaptionvolstructure.hpp>`

Inheritance diagram for SwaptionVolatilityStructure:



11.369.1 Detailed Description

Swaption volatility structure.

This class is purely abstract and defines the interface of concrete swaption volatility structures which will be derived from this one.

Public Member Functions

- virtual **Date todaysDate** () const =0
  returns today's date
- virtual **DayCounter dayCounter** () const =0
  returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &start, const **Period** &length, **Rate** strike) const
  returns the volatility for a given starting date and length
- double **volatility** (**Time** start, **Time** length, **Rate** strike) const
  returns the volatility for a given starting time and length

Protected Member Functions

- virtual double **volatilityImpl** (**Time** start, **Time** length, **Rate** strike) const =0
  implements the actual volatility calculation in derived classes
- virtual std::pair< **Time**, **Time** > **convertDates** (const **Date** &start, const **Period** &length) const
  implements the conversion between dates and times

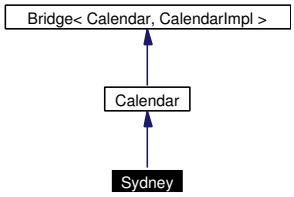
The documentation for this class was generated from the following file:

- ql/**swaptionvolstructure.hpp**

11.370 Sydney Class Reference

`#include <sydney.hpp>`

Inheritance diagram for Sydney:



11.370.1 Detailed Description

Sydney, calendar (New South Wales, Australia)

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Australia Day, January 26th (possibly moved to Monday)
- Good Friday
- Easter Monday
- ANZAC Day. April 25th (possibly moved to Monday)
- Queen's Birthday, second Monday in June
- Bank Holiday, first Monday in August
- Labour Day, first Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

The documentation for this class was generated from the following file:

- ql/Calendars/**sydney.hpp**

11.371 SymmetricSchurDecomposition Class Reference

`#include <symmetricschurdecomposition.hpp>`

11.371.1 Detailed Description

symmetric threshold Jacobi algorithm.

Given a real symmetric matrix S, the Schur decomposition finds the eigenvalues and eigenvectors of S. If D is the diagonal matrix formed by the eigenvalues and U the unitarian matrix of the eigenvectors we can write the Schur decomposition as

$$S = U \cdot D \cdot U^T ,$$

where $\cdot$ is the standard matrix product and ^{T} is the transpose operator. This class implements the Schur decomposition using the symmetric threshold Jacobi algorithm. For details on the different Jacobi transfomations you can start from the great book on matrix computations by Golub and Van Loan: **Matrix**(p. 471) computation, second edition The Johns Hopkins University Press

Public Member Functions

- **SymmetricSchurDecomposition** (const **Matrix** &s)
- const **Array** & **eigenvalues** () const
- const **Matrix** & **eigenvectors** () const

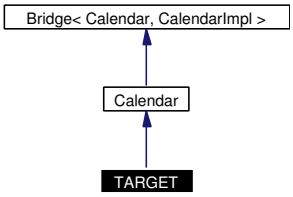
The documentation for this class was generated from the following files:

- ql/Math/**symmetricschurdecomposition.hpp**
- ql/Math/**symmetricschurdecomposition.cpp**

11.372 TARGET Class Reference

`#include <target.hpp>`

Inheritance diagram for TARGET:



11.372.1 Detailed Description

TARGET calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday (since 2000)
- Easter Monday (since 2000)
- Labour Day, May 1st (since 2000)
- Christmas, December 25th
- Day of Goodwill, December 26th (since 2000)
- December 31st (1998, 1999, and 2001)

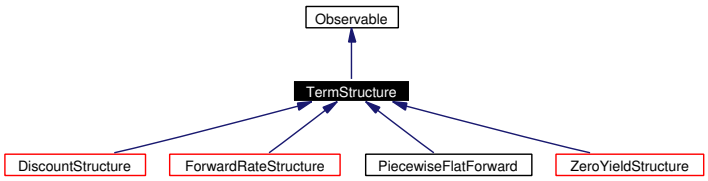
The documentation for this class was generated from the following file:

- ql/Calendars/**target.hpp**

11.373 TermStructure Class Reference

`#include <termstructure.hpp>`

Inheritance diagram for TermStructure:



11.373.1 Detailed Description

Term structure.

This abstract class defines the interface of concrete rate structures which will be derived from this one.

Rates are assumed to be annual continuos compounding.

Todo

add derived class ParSwapTermStructure similar to ZeroYieldTermStructure, **DiscountStructure**(p. 303), **ForwardRateStructure**(p. 361)
allow for different compounding rules and compounding frequencies

Public Member Functions

Rates and discount

These methods are either function of dates or times. In the latter case, times are calculated as fraction of year from the reference date.

- **Rate zeroYield** (const **Date** &, bool extrapolate=false) const
  zero-yield rate
- **Rate zeroYield** (**Time**, bool extrapolate=false) const
  zero-yield rate
- **DiscountFactor discount** (const **Date** &, bool extrapolate=false) const
  discount factor
- **DiscountFactor discount** (**Time**, bool extrapolate=false) const
  discount factor
- **Rate instantaneousForward** (const **Date** &, bool extrapolate=false) const
  instantaneous forward rate
- **Rate instantaneousForward** (**Time**, bool extrapolate=false) const
  instantaneous forward rate

- **Rate compoundForward** (const **Date** &, int, bool extrapolate=false) const
  instantaneous forward rate at a given compounding frequency
- **Rate compoundForward** (**Time**, int, bool extrapolate=false) const
  instantaneous forward rate at a given compounding frequency
- **Rate forward** (const **Date** &, const **Date** &, bool extrapolate=false) const
  discrete forward rate between two dates
- **Rate forward** (**Time**, **Time**, bool extrapolate=false) const
  discrete forward rate between two times
- **Rate zeroCoupon** (const **Date** &, int, bool extrapolate=false) const
  zero-coupon rate
- **Rate zeroCoupon** (**Time**, int, bool extrapolate=false) const
  zero-coupon rate

Dates

- virtual **Date todaysDate** () const =0
  today's date
- virtual **Date referenceDate** () const =0
  the reference date, i.e., the date at which discount = 1
- virtual **DayCounter dayCounter** () const =0
  the day counter used for date/time conversion
- virtual **Date maxDate** () const =0
  the latest date for which the curve can return rates
- virtual **Time maxTime** () const
  the latest time for which the curve can return rates

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual discount and rate calculations

- virtual **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const =0
  zero-yield calculation
- virtual **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const =0
  discount calculation
- virtual **Rate forwardImpl** (**Time**, bool extrapolate=false) const =0
  instantaneous forward-rate calculation

- virtual **Rate compoundForwardImpl** (**Time**, int, bool extrapolate=false) const =0
  compound forward-rate calculation

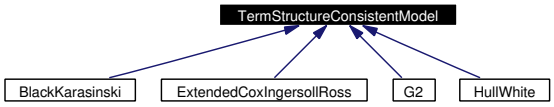
The documentation for this class was generated from the following file:

- ql/**termstructure.hpp**

11.374 TermStructureConsistentModel Class Reference

`#include <model.hpp>`

Inheritance diagram for TermStructureConsistentModel:



11.374.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

Public Member Functions

- **TermStructureConsistentModel** (const **RelinkableHandle**< **TermStructure** > &termStructure)
- const **RelinkableHandle**< **TermStructure** > & **termStructure** () const

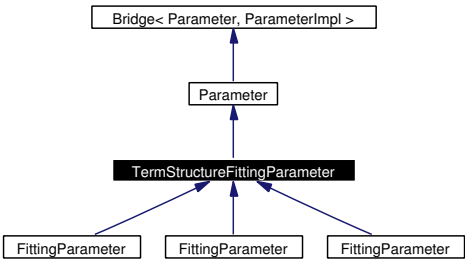
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**model.hpp**

11.375 TermStructureFittingParameter Class Reference

`#include <parameter.hpp>`

Inheritance diagram for TermStructureFittingParameter:



11.375.1 Detailed Description

Deterministic time-dependent parameter used for yield-curve fitting.

Public Member Functions

- **TermStructureFittingParameter** (const **Handle**< Parameter::Impl > &impl)
- **TermStructureFittingParameter** (const **RelinkableHandle**< **TermStructure** > &term)

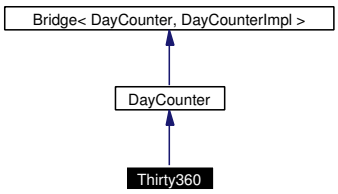
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**parameter.hpp**

11.376 Thirty360 Class Reference

`#include <thirty360.hpp>`

Inheritance diagram for Thirty360:



11.376.1 Detailed Description

30/360 day count convention

The day count can be calculated according to US, European, or Italian conventions.

US (NASD) convention: if the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month.

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month.

Italian convention: starting dates or ending dates that occur on February and are grater than 27 become equal to 30 for computational sake.

Public Types

- enum **Convention** { **USA**, **European**, **Italian** }

Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

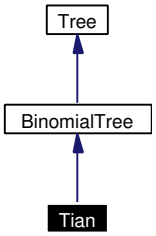
The documentation for this class was generated from the following files:

- ql/DayCounters/**thirty360.hpp**
- ql/DayCounters/**thirty360.cpp**

11.377 Tian Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for Tian:



11.377.1 Detailed Description

Tian(p. 640) tree: third moment matching, multiplicative approach.

Public Member Functions

- **Tian** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)
- double **underlying** (**Size** i, **Size** index) const
- double **probability** (**Size**, **Size**, **Size**) const

Protected Attributes

- double **up_**
- double **down_**
- double **pu_**
- double **pd_**

The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.378 TimeBasket Class Reference

`#include <timebasket.hpp>`

11.378.1 Detailed Description

Distribution over a number of dates.

Map interface

- typedef super::iterator **iterator**
- typedef super::const_iterator **const_iterator**
- typedef super::reverse_iterator **reverse_iterator**
- typedef super::const_reverse_iterator **const_reverse_iterator**
- bool **hasDate** (const **Date** &) const

  membership

Public Member Functions

- **TimeBasket** (const std::vector< **Date** > &dates, const std::vector< double > &values)

Algebra

- **TimeBasket** & **operator+=** (const **TimeBasket** &other)
- **TimeBasket** & **operator-=** (const **TimeBasket** &other)

Other methods

- **TimeBasket rebin** (const std::vector< **Date** > &buckets) const

  redistribute the entries over the given dates

The documentation for this class was generated from the following files:

- ql/CashFlows/**timebasket.hpp**
- ql/CashFlows/**timebasket.cpp**

11.379 TimeGrid Class Reference

`#include <grid.hpp>`

11.379.1 Detailed Description

time grid class

Todo

What was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Public Member Functions

- **TimeGrid** (**Time** end, **Size** steps)
  Regularly spaced time-grid.
- template<class Iterator> **TimeGrid** (Iterator begin, Iterator end)
  Time grid with mandatory time points.
- template<class Iterator> **TimeGrid** (Iterator begin, Iterator end, **Size** steps)
  Time grid with mandatory time points.
- **Size findIndex** (**Time** t) const
- const std::vector< **Time** > & **mandatoryTimes** () const
- **Time dt** (**Size** i) const

11.379.2 Constructor & Destructor Documentation

11.379.2.1 TimeGrid (Iterator *begin*, Iterator *end*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. No additional points are added.

11.379.2.2 TimeGrid (Iterator *begin*, Iterator *end*, Size *steps*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. Additional points are then added with regular spacing between pairs of mandatory times in order to reach the desired number of steps.

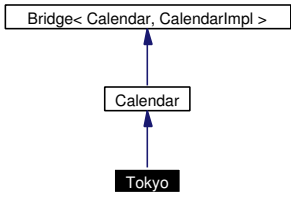
The documentation for this class was generated from the following files:

- ql/**grid.hpp**
- ql/**grid.cpp**

11.380 Tokyo Class Reference

`#include <tokyo.hpp>`

Inheritance diagram for Tokyo:



11.380.1 Detailed Description

Tokyo calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Bank Holiday, January 2nd
- Bank Holiday, January 3rd
- Coming of Age Day, 2nd Monday in January
- National Foundation Day, February 11th
- Vernal Equinox
- Greenery Day, April 29th
- Constitution Memorial Day, May 3rd
- Holiday for a Nation, May 4th
- Children's Day, May 5th
- Marine Day, 3rd Monday in July
- Respect for the Aged Day, 3rd Monday in September
- Autumnal Equinox
- Health and Sports Day, 2nd Monday in October
- National Culture Day, November 3rd
- Labor Thanksgiving Day, November 23rd
- Emperor's Birthday, December 23rd
- Bank Holiday, December 31st

- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the Bank Holidays associated with the New Year

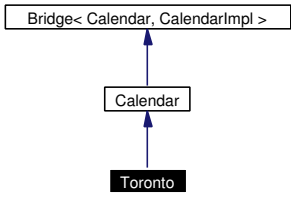
The documentation for this class was generated from the following file:

- ql/Calendars/**tokyo.hpp**

11.381 Toronto Class Reference

`#include <toronto.hpp>`

Inheritance diagram for Toronto:



11.381.1 Detailed Description

Toronto calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Victoria Day, The Monday on or preceding 24 May
- Canada Day, July 1st (possibly moved to Monday)
- Provincial Holiday, first Monday of August
- Labour Day, first Monday of September
- Thanksgiving Day, second Monday of October
- Remembrance Day, November 11th
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

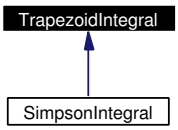
The documentation for this class was generated from the following file:

- ql/Calendars/**toronto.hpp**

11.382 TrapezoidIntegral Class Reference

`#include <trapezoidintegral.hpp>`

Inheritance diagram for TrapezoidIntegral:



11.382.1 Detailed Description

Integral of a one-dimensional function.

Given a target accuracy ϵ, the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f\,\mathrm{d}x = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b-a)/N$. The number N of intervals is repeatedly increased until the target accuracy is reached.

Public Types

- enum **Method** { **Default**, **MidPoint** }

Public Member Functions

- **TrapezoidIntegral** (double accuracy, Method method=Default, **Size** maxIterations=**Null**< int >())
- template<class F> double **operator()** (const F &f, double a, double b) const
- double **accuracy** () const
- double & **accuracy** ()
- Method **method** () const
- Method & **method** ()
- **Size maxIterations** () const
- **Size** & **maxIterations** ()

Protected Member Functions

- template<class F> double **defaultIteration** (const F &f, double a, double b, double I, **Size** N) const
- template<class F> double **midPointIteration** (const F &f, double a, double b, double I, **Size** N) const

Protected Attributes

- double **accuracy_**
- Method **method_**
- **Size maxIterations_**

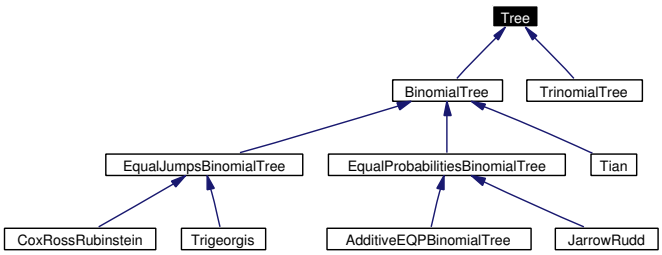
The documentation for this class was generated from the following file:

- ql/Math/**trapezoidintegral.hpp**

11.383 Tree Class Reference

`#include <tree.hpp>`

Inheritance diagram for Tree:



11.383.1 Detailed Description

Tree(p. 648) approximating a single-factor diffusion.

Public Member Functions

- **Tree** (**Size** nColumns)
- virtual double **underlying** (**Size** i, **Size** index) const =0
- virtual **Size size** (**Size** i) const =0
- virtual **Size descendant** (**Size** i, **Size** index, **Size** branch) const =0
- virtual double **probability** (**Size** i, **Size** index, **Size** branch) const =0
- **Size nColumns** () const

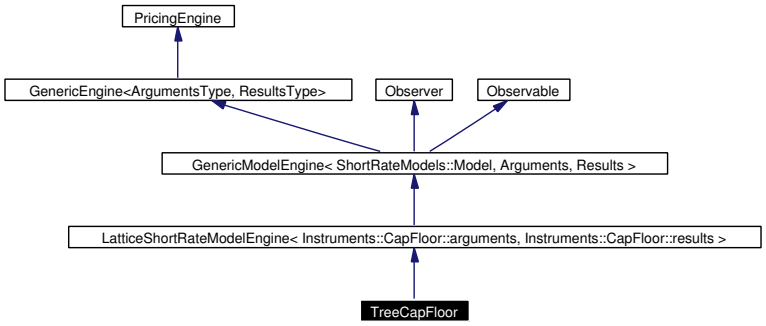
The documentation for this class was generated from the following file:

- ql/Lattices/**tree.hpp**

11.384 TreeCapFloor Class Reference

`#include <treecapfloor.hpp>`

Inheritance diagram for TreeCapFloor:



11.384.1 Detailed Description

Cap/Floor priced in a tree.

Public Member Functions

- **TreeCapFloor** (const **Handle**< **ShortRateModels::Model** > &model, **Size** timeSteps)
- **TreeCapFloor** (const **Handle**< **ShortRateModels::Model** > &model, const **TimeGrid** &timeGrid)
- void **calculate** () const

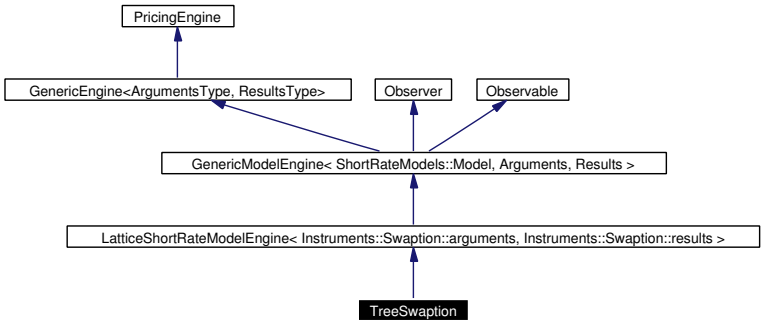
The documentation for this class was generated from the following files:

- ql/Pricers/**treecapfloor.hpp**
- ql/Pricers/**treecapfloor.cpp**

11.385 TreeSwaption Class Reference

`#include <treeswaption.hpp>`

Inheritance diagram for TreeSwaption:



11.385.1 Detailed Description

Swaption priced on a lattice.

Public Member Functions

- **TreeSwaption** (const **Handle**< **ShortRateModels::Model** > &model, **Size** timeSteps)
- **TreeSwaption** (const **Handle**< **ShortRateModels::Model** > &model, const **TimeGrid** &timeGrid)
- void **calculate** () const

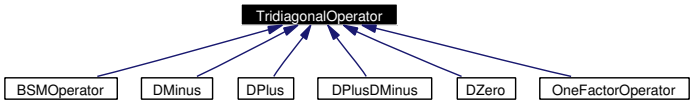
The documentation for this class was generated from the following files:

- ql/Pricers/**treeswaption.hpp**
- ql/Pricers/**treeswaption.cpp**

11.386 TridiagonalOperator Class Reference

`#include <tridiagonaloperator.hpp>`

Inheritance diagram for TridiagonalOperator:



11.386.1 Detailed Description

Base implementation for tridiagonal operator.

Warning:
to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class

Operator interface

- **Disposable**< **Array** > **applyTo** (const **Array** &v) const
  apply operator to a given array
- **Disposable**< **Array** > **solveFor** (const **Array** &rhs) const
  solve linear system for a given right-hand side
- **Disposable**< **Array** > **SOR** (const **Array** &rhs, double tol) const
  solve linear system with SOR approach
- **Disposable**< **TridiagonalOperator** > **identity** (**Size** size)
  identity instance

Public Types

- typedef **Array** **arrayType**

Public Member Functions

- **TridiagonalOperator** (**Size** size=0)
- **TridiagonalOperator** (const **Array** &low, const **Array** &mid, const **Array** &high)
- **TridiagonalOperator** (const **Disposable**< **TridiagonalOperator** > &)
- **TridiagonalOperator** & **operator=** (const **Disposable**< **TridiagonalOperator** > &)

Inspectors

- **Size** **size** () const

- bool **isTimeDependent** ()
- const **Array** & **lowerDiagonal** () const
- const **Array** & **diagonal** () const
- const **Array** & **upperDiagonal** () const

Modifiers

- void **setFirstRow** (double, double)
- void **setMidRow** (**Size**, double, double, double)
- void **setMidRows** (double, double, double)
- void **setLastRow** (double, double)
- void **setTime** (**Time** t)

Utilities

- void **swap** (**TridiagonalOperator** &)

Protected Attributes

- **Array diagonal_**
- **Array lowerDiagonal_**
- **Array upperDiagonal_**
- **Handle**< **TimeSetter** > **timeSetter_**

Friends

- **Disposable**< **TridiagonalOperator** > **operator+** (const **TridiagonalOperator** &)
- **Disposable**< **TridiagonalOperator** > **operator-** (const **TridiagonalOperator** &)
- **Disposable**< **TridiagonalOperator** > **operator+** (const **TridiagonalOperator** &, const **TridiagonalOperator** &)
- **Disposable**< **TridiagonalOperator** > **operator-** (const **TridiagonalOperator** &, const **TridiagonalOperator** &)
- **Disposable**< **TridiagonalOperator** > **operator** * (double, const **TridiagonalOperator** &)
- **Disposable**< **TridiagonalOperator** > **operator** * (const **TridiagonalOperator** &, double)
- **Disposable**< **TridiagonalOperator** > **operator/** (const **TridiagonalOperator** &, double)

The documentation for this class was generated from the following files:

- ql/FiniteDifferences/**tridiagonaloperator.hpp**
- ql/FiniteDifferences/**tridiagonaloperator.cpp**

11.387 TridiagonalOperator::TimeSetter Class Reference

`#include <tridiagonaloperator.hpp>`

11.387.1 Detailed Description

encapsulation of time-setting logic

Public Member Functions

- virtual void **setTime** (**Time** t, **TridiagonalOperator** &L) const =0

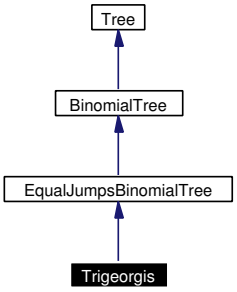
The documentation for this class was generated from the following file:

- ql/FiniteDifferences/**tridiagonaloperator.hpp**

11.388 Trigeorgis Class Reference

`#include <binomialtree.hpp>`

Inheritance diagram for Trigeorgis:



11.388.1 Detailed Description

Trigeorgis(p. 654) (additive equal jumps) binomial tree.

Public Member Functions

- **Trigeorgis** (const **Handle**< **DiffusionProcess** > &process, **Time** end, **Size** steps)

The documentation for this class was generated from the following files:

- ql/Lattices/**binomialtree.hpp**
- ql/Lattices/**binomialtree.cpp**

11.389 TrinomialBranching Class Reference

`#include <trinomialtree.hpp>`

11.389.1 Detailed Description

Branching scheme for a trinomial node.

Each node has three descendants, with the middle branch linked to the node which is closest to the expectation of the variable.

Public Member Functions

- **Size descendant** (**Size** index, **Size** branch) const
- double **probability** (**Size** index, **Size** branch) const
- int **jMin** () const

Friends

- class **TrinomialTree**

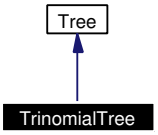
The documentation for this class was generated from the following file:

- ql/Lattices/**trinomialtree.hpp**

11.390 TrinomialTree Class Reference

`#include <trinomialtree.hpp>`

Inheritance diagram for TrinomialTree:



11.390.1 Detailed Description

Recombining trinomial tree class.

This class defines a recombining trinomial tree approximating a a diffusion.

Warning:
The diffusion term of the SDE must be independent of the underlying process.

Public Member Functions

- **TrinomialTree** (const **Handle**< **DiffusionProcess** > &process, const **TimeGrid** &timeGrid, bool isPositive=false)
- double **dx** (**Size** i) const
- **Size size** (**Size** i) const
- double **underlying** (**Size** i, **Size** index) const
- const **TimeGrid** & **timeGrid** () const
- **Size descendant** (**Size** i, **Size** index, **Size** branch) const
- double **probability** (**Size** i, **Size** index, **Size** branch) const

Protected Attributes

- std::vector< **Handle**< **TrinomialBranching** > > **branchings_**
- double **x0_**
- std::vector< double > **dx_**
- **TimeGrid timeGrid_**

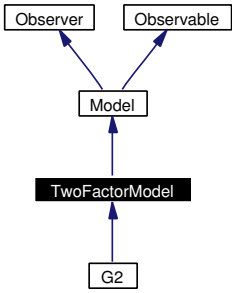
The documentation for this class was generated from the following files:

- ql/Lattices/**trinomialtree.hpp**
- ql/Lattices/**trinomialtree.cpp**

11.391 TwoFactorModel Class Reference

`#include <twofactormodel.hpp>`

Inheritance diagram for TwoFactorModel:



11.391.1 Detailed Description

Abstract base-class for two-factor models.

Public Member Functions

- **TwoFactorModel** (**Size** nParams)
- virtual **Handle**< **ShortRateDynamics** > **dynamics** () const =0

  Returns the short-rate dynamics.

- virtual **Handle**< **Lattices::Lattice** > **tree** (const **TimeGrid** &grid) const

  Returns a two-dimensional trinomial tree.

The documentation for this class was generated from the following files:

- ql/ShortRateModels/**twofactormodel.hpp**
- ql/ShortRateModels/**twofactormodel.cpp**

11.392 TwoFactorModel::ShortRateDynamics Class Reference

`#include <twofactormodel.hpp>`

11.392.1 Detailed Description

Class describing the dynamics of the two state variables.

We assume here that the short-rate is a function of two state variables x and y.

$$r_t = f(t, x_t, y_t)$$

of two state variables x_t and y_t. These stochastic processes satisfy

$$x_t = \mu_x(t, x_t)dt + \sigma_x(t, x_t)dW_t^x$$

and

$$y_t = \mu_y(t, y_t)dt + \sigma_y(t, y_t)dW_t^y$$

where W^x and W^y are two brownian motions satisfying

$$dW_t^x dW_t^y = \rho dt$$

.

Public Member Functions

- **ShortRateDynamics** (const **Handle**< **DiffusionProcess** > &xProcess, const **Handle**< **DiffusionProcess** > &yProcess, double correlation)
- virtual **Rate shortRate** (**Time** t, double x, double y) const =0
- const **Handle**< **DiffusionProcess** > & **xProcess** () const

  Risk-neutral dynamics of the first state variable x.

- const **Handle**< **DiffusionProcess** > & **yProcess** () const

  Risk-neutral dynamics of the second state variable y.

- double **correlation** () const

  Correlation ρ between the two brownian motions.

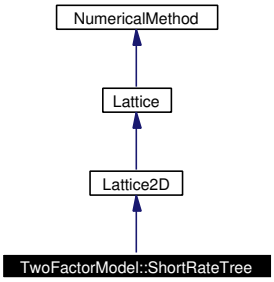
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**twofactormodel.hpp**

11.393 TwoFactorModel::ShortRateTree Class Reference

`#include <twofactormodel.hpp>`

Inheritance diagram for TwoFactorModel::ShortRateTree:



11.393.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

Public Member Functions

- **ShortRateTree** (const **Handle**< **Lattices::TrinomialTree** > &tree1, const **Handle**< **Lattices::TrinomialTree** > &tree2, const **Handle**< **ShortRateDynamics** > &dynamics)

  Plain tree build-up from short-rate dynamics.

- **DiscountFactor discount** (**Size** i, **Size** index) const

  Discount factor at time t_i and node indexed by index.

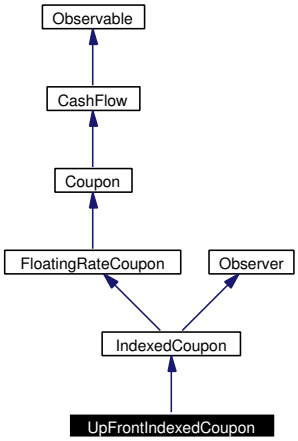
The documentation for this class was generated from the following file:

- ql/ShortRateModels/**twofactormodel.hpp**

11.394 UpFrontIndexedCoupon Class Reference

`#include <upfrontindexedcoupon.hpp>`

Inheritance diagram for UpFrontIndexedCoupon:



11.394.1 Detailed Description

up front indexed coupon class

Warning:
This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **UpFrontIndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Indexes::Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, **Spread** spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

FloatingRateCoupon interface

- **Date fixingDate** () const

Visitability

- virtual void **accept** (**Patterns::AcyclicVisitor** &)

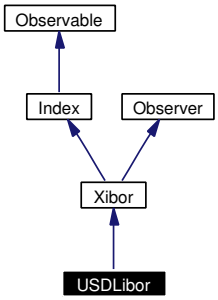
The documentation for this class was generated from the following file:

- ql/CashFlows/**upfrontindexedcoupon.hpp**

11.395 USDLibor Class Reference

`#include <usdlibor.hpp>`

Inheritance diagram for USDLibor:



11.395.1 Detailed Description

USD Libor index

Public Member Functions

- **USDLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual360**())

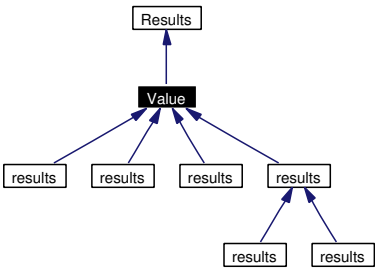
The documentation for this class was generated from the following file:

- ql/Indexes/**usdlibor.hpp**

11.396 Value Class Reference

`#include <instrument.hpp>`

Inheritance diagram for Value:



11.396.1 Detailed Description

pricing results

Public Member Functions

- void **reset** ()

Public Attributes

- double **value**
- double **errorEstimate**

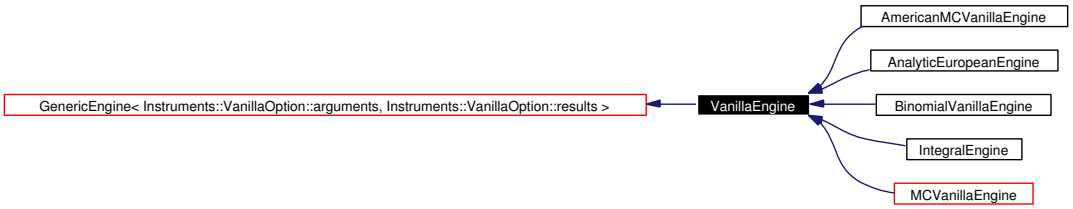
The documentation for this class was generated from the following file:

- ql/**instrument.hpp**

11.397 VanillaEngine Class Reference

`#include <vanillaengines.hpp>`

Inheritance diagram for VanillaEngine:



11.397.1 Detailed Description

Vanilla engine base class.

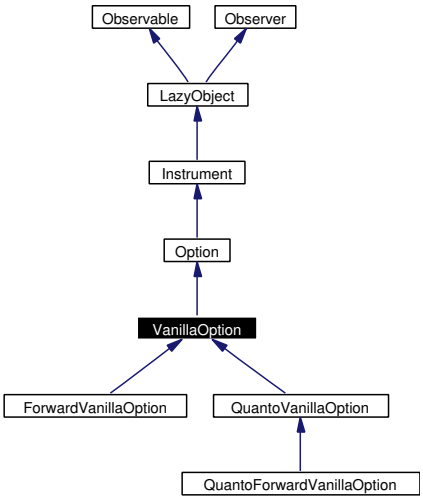
The documentation for this class was generated from the following file:

- ql/PricingEngines/**vanillaengines.hpp**

11.398 VanillaOption Class Reference

`#include <vanillaoption.hpp>`

Inheritance diagram for VanillaOption:



11.398.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

Public Member Functions

- **VanillaOption** (Option::Type type, const **RelinkableHandle**< **MarketElement** > &underlying, double strike, const **RelinkableHandle**< **TermStructure** > &dividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **Exercise** &exercise, const **RelinkableHandle**< **BlackVolTermStructure** > &volTS, const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >(), const std::string &isinCode="", const std::string &description="")
- double **impliedVolatility** (double price, double accuracy=1.0e-4, **Size** maxEvaluations=100, double minVol=QL_MIN_VOLATILITY, double maxVol=QL_MAX_VOLATILITY) const
- void **setupArguments** (**Arguments** *) const

Instrument interface

- bool **isExpired** () const

  returns whether the instrument is still tradable.

greeks

- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const

- double **rho** () const
- double **dividendRho** () const
- double **strikeSensitivity** () const

Protected Member Functions

- void **setupExpired** () const
- void **performCalculations** () const

Protected Attributes

- double **delta_**
- double **gamma_**
- double **theta_**
- double **vega_**
- double **rho_**
- double **dividendRho_**
- double **strikeSensitivity_**
- Option::Type **type_**
- **RelinkableHandle**< **MarketElement** > **underlying_**
- double **strike_**
- **Exercise exercise_**
- **RelinkableHandle**< **TermStructure** > **riskFreeTS_**
- **RelinkableHandle**< **TermStructure** > **dividendTS_**
- **RelinkableHandle**< **BlackVolTermStructure** > **volTS_**

11.398.2 Member Function Documentation

11.398.2.1 double impliedVolatility (double *price*, double *accuracy* = 1.0e-4, Size *maxEvaluations* = 100, double *minVol* = QL_MIN_VOLATILITY, double *maxVol* = QL_MAX_VOLATILITY) const

Warning:

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases impliedVolatility can fail and in any case it is almost meaningless. Another possible source of failure is to have a targetValue that is not attainable with any volatility, e.g. a targetValue lower than the intrinsic value in the case of American options.

11.398.2.2 void setupArguments (Arguments *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. 423).

Reimplemented in **ForwardVanillaOption** (p. 366), **QuantoForwardVanillaOption** (p. 567), and **QuantoVanillaOption** (p. 573).

11.398.2.3 void setupExpired () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from **Instrument** (p. 424).

Reimplemented in **QuantoVanillaOption** (p. 573).

11.398.2.4 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. 424).

Reimplemented in **ForwardVanillaOption** (p. 366), and **QuantoVanillaOption** (p. 573).

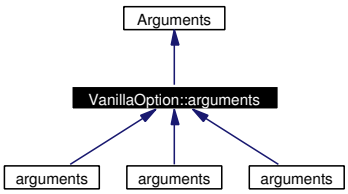
The documentation for this class was generated from the following files:

- ql/Instruments/**vanillaoption.hpp**
- ql/Instruments/**vanillaoption.cpp**

11.399 VanillaOption::arguments Class Reference

`#include <vanillaoption.hpp>`

Inheritance diagram for VanillaOption::arguments:



11.399.1 Detailed Description

arguments for vanilla option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- **Handle**< **Payoff** > **payoff**
- double **underlying**
- **RelinkableHandle**< **TermStructure** > **riskFreeTS**
- **RelinkableHandle**< **TermStructure** > **dividendTS**
- **RelinkableHandle**< **BlackVolTermStructure** > **volTS**
- Exercise::Type **exerciseType**
- std::vector< **Time** > **stoppingTimes**
- **Time maturity**

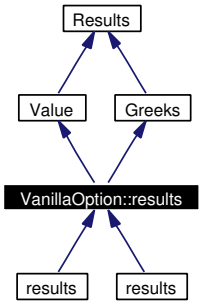
The documentation for this class was generated from the following files:

- ql/Instruments/**vanillaoption.hpp**
- ql/Instruments/**vanillaoption.cpp**

11.400 VanillaOption::results Class Reference

`#include <vanillaoption.hpp>`

Inheritance diagram for VanillaOption::results:



11.400.1 Detailed Description

results from vanilla option calculation

Public Member Functions

- void **reset** ()

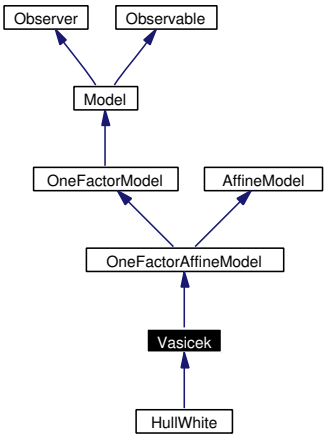
The documentation for this class was generated from the following file:

- ql/Instruments/**vanillaoption.hpp**

11.401 Vasicek Class Reference

`#include <vasicek.hpp>`

Inheritance diagram for Vasicek:



11.401.1 Detailed Description

Vasicek(p. 669) model class.

This class implements the **Vasicek**(p. 669) model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where a, b and σ are constants.

Public Member Functions

- **Vasicek** (**Rate** r0=0.05, double a=0.1, double b=0.05, double sigma=0.01)
- virtual double **discountBondOption** (Option::Type type, double strike, **Time** maturity, **Time** bondMaturity) const
- virtual **Handle**< ShortRateDynamics > **dynamics** () const

  returns the short-rate dynamics

Protected Member Functions

- virtual double **A** (**Time** t, **Time** T) const
- virtual double **B** (**Time** t, **Time** T) const
- double **a** () const
- double **b** () const
- double **sigma** () const

The documentation for this class was generated from the following files:

- ql/ShortRateModels/OneFactorModels/**vasicek.hpp**
- ql/ShortRateModels/OneFactorModels/**vasicek.cpp**

11.402 Vasicek::Dynamics Class Reference

`#include <vasicek.hpp>`

11.402.1 Detailed Description

Short-rate dynamics in the **Vasicek**(p. 669) model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b.

Public Member Functions

- **Dynamics** (double a, double b, double sigma, double r0)
- virtual double **variable** (**Time** t, **Rate** r) const
- virtual double **shortRate** (**Time** t, double x) const

The documentation for this class was generated from the following file:

- ql/ShortRateModels/OneFactorModels/**vasicek.hpp**

11.403 Visitor Class Template Reference

`#include <visitor.hpp>`

11.403.1 Detailed Description

template<class T> class QuantLib::Patterns::Visitor< T >

visitor for a specific class

Public Member Functions

- virtual void **visit** (T &)=0

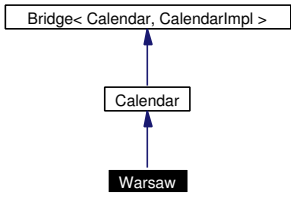
The documentation for this class was generated from the following file:

- ql/Patterns/**visitor.hpp**

11.404 Warsaw Class Reference

`#include <warsaw.hpp>`

Inheritance diagram for Warsaw:



11.404.1 Detailed Description

Warsaw calendar

Holidays:

- Saturdays
- Sundays
- Easter Monday
- Corpus Christi
- New Year's Day, January 1st
- May Day, May 1st
- Constitution Day, May 3rd
- Assumption of the Blessed Virgin Mary, August 15th
- All Saints Day, November 1st
- Independence Day, November 11th
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

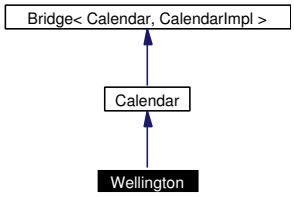
The documentation for this class was generated from the following file:

- ql/Calendars/**warsaw.hpp**

11.405 Wellington Class Reference

`#include <wellington.hpp>`

Inheritance diagram for Wellington:



11.405.1 Detailed Description

Wellington calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday or Tuesday)
- Day after New Year's Day, January 2st (possibly moved to Monday or Tuesday)
- Anniversary Day, Monday nearest January 22nd
- Waitangi Day. February 6th
- Good Friday
- Easter Monday
- ANZAC Day. April 25th
- Queen's Birthday, first Monday in June
- Labour Day, fourth Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Note:
The holiday rules for **Wellington**(p. 674) were documented by David Gilbert for IDB (`http://www.jrefinery.com/ibd/`)

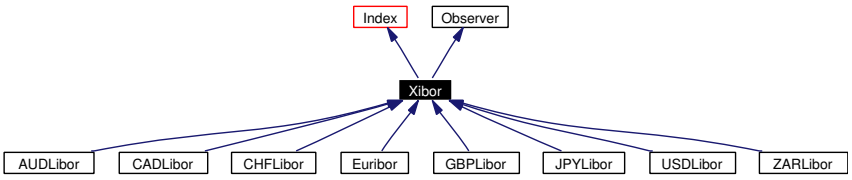
The documentation for this class was generated from the following file:

- ql/Calendars/**wellington.hpp**

11.406 Xibor Class Reference

`#include <xibor.hpp>`

Inheritance diagram for Xibor:



11.406.1 Detailed Description

base class for libor indexes

Public Member Functions

- **Xibor** (const std::string &familyName, int n, **TimeUnit** units, int settlementDays, **Currency** currency, const **Calendar** &calendar, bool isAdjusted, **RollingConvention** rollingConvention, const **DayCounter** &dayCounter, const **RelinkableHandle**< **TermStructure** > &h)

Index interface

- **Rate fixing** (const **Date** &fixingDate) const
  returns the fixing at the given date

Observer interface

- void **update** ()

Inspectors

- std::string **name** () const
  Returns the name of the index.
- **Period tenor** () const
- int **frequency** () const
- int **settlementDays** () const
- **Currency currency** () const
- **Calendar calendar** () const
- bool **isAdjusted** () const
- **RollingConvention rollingConvention** () const
- **DayCounter dayCounter** () const
- **Handle**< **TermStructure** > **termStructure** () const

11.406.2 Member Function Documentation

11.406.2.1 Rate fixing (const Date & *fixingDate*) const [virtual]

returns the fixing at the given date

Note:
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implements **Index** (p. 417).

11.406.2.2 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.406.2.3 std::string name () const [virtual]

Returns the name of the index.

Warning:
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements **Index** (p. 417).

The documentation for this class was generated from the following files:

- ql/Indexes/**xibor.hpp**
- ql/Indexes/**xibor.cpp**

11.407 XiborManager Class Reference

`#include <xibormanager.hpp>`

11.407.1 Detailed Description

global repository for libor histories

Public Types

- typedef std::map< std::string, **History** > **HistoryMap**

Static Public Member Functions

- void **setHistory** (const std::string &name, const **History** &)
- const **History** & **getHistory** (const std::string &name)
- bool **hasHistory** (const std::string &name)
- std::vector< std::string > **histories** ()

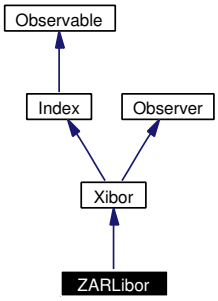
The documentation for this class was generated from the following files:

- ql/Indexes/**xibormanager.hpp**
- ql/Indexes/**xibormanager.cpp**

11.408 ZARLibor Class Reference

`#include <zarlibor.hpp>`

Inheritance diagram for ZARLibor:



11.408.1 Detailed Description

ZAR Libor index (also known as JIBAR)

Todo
check settlement days

Public Member Functions

- **ZARLibor** (int n, **TimeUnit** units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**DayCounters::Actual365**())

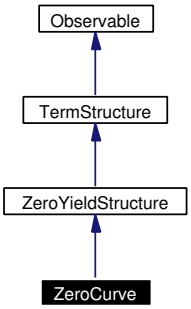
The documentation for this class was generated from the following file:

- ql/Indexes/**zarlibor.hpp**

11.409 ZeroCurve Class Reference

`#include <zerocurve.hpp>`

Inheritance diagram for ZeroCurve:



11.409.1 Detailed Description

Term structure based on linear interpolation of zero yields.

Public Member Functions

- **ZeroCurve** (const **Date** &todaysDate, const std::vector< **Date** > &dates, const std::vector< **Rate** > &yields, const **DayCounter** &dayCounter=**DayCounters::Actual365**())
- **DayCounter dayCounter** () const

  the day counter used for date/time conversion

- **Date todaysDate** () const

  today's date

- **Date referenceDate** () const

  the reference date, i.e., the date at which discount = 1

- const std::vector< **Date** > & **dates** () const
- **Date maxDate** () const

  the latest date for which the curve can return rates

- const std::vector< **Time** > & **times** () const
- **Time maxTime** () const

  the latest time for which the curve can return rates

Protected Member Functions

- **Rate zeroYieldImpl** (**Time** t, bool extrapolate=false) const

  zero-yield calculation

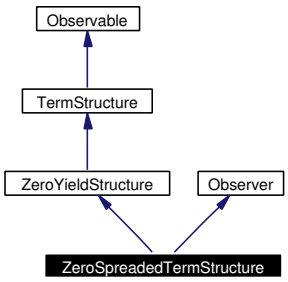
The documentation for this class was generated from the following files:

- ql/TermStructures/**zerocurve.hpp**
- ql/TermStructures/**zerocurve.cpp**

11.410 ZeroSpreadedTermStructure Class Reference

`#include <zerospreadedtermstructure.hpp>`

Inheritance diagram for ZeroSpreadedTermStructure:



11.410.1 Detailed Description

Term structure with an added spread on the zero yield rate.

Note:
This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Public Member Functions

- **ZeroSpreadedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **RelinkableHandle**< **MarketElement** > &spread)

TermStructure interface

- **DayCounter dayCounter** () const
  the day counter used for date/time conversion
- **Date referenceDate** () const
  the reference date, i.e., the date at which discount = 1
- **Date todaysDate** () const
  today's date
- **Date maxDate** () const
  the latest date for which the curve can return rates
- **Time maxTime** () const
  the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- **Rate zeroYieldImpl** (**Time**, bool extrapolate=false) const
  returns the spreaded zero yield rate
- **Rate forwardImpl** (**Time**, bool extrapolate=false) const
  returns the spreaded forward rate

11.410.2 Member Function Documentation

11.410.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 527).

11.410.2.2 Rate forwardImpl (Time, bool *extrapolate* = false) const [protected, virtual]

returns the spreaded forward rate

Warning:
This method must disappear should the spread become a curve

Reimplemented from **ZeroYieldStructure** (p. 683).

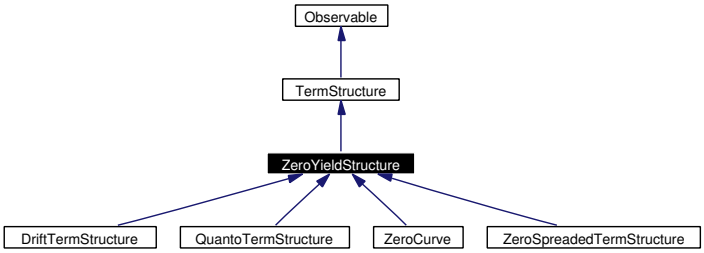
The documentation for this class was generated from the following file:

- ql/TermStructures/**zerospreadedtermstructure.hpp**

11.411 ZeroYieldStructure Class Reference

`#include <termstructure.hpp>`

Inheritance diagram for ZeroYieldStructure:



11.411.1 Detailed Description

Zero yield term structure.

This abstract class acts as an adapter to **TermStructure**(p. 634) allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes.

Rates are assumed to be annual continuos compounding.

Protected Member Functions

- **DiscountFactor discountImpl** (**Time**, bool extrapolate=false) const
- **Rate forwardImpl** (**Time**, bool extrapolate=false) const
- **Rate compoundForwardImpl** (**Time**, int, bool extrapolate=false) const

11.411.2 Member Function Documentation

11.411.2.1 DiscountFactor discountImpl (Time, bool *extrapolate* = false) const [protected, virtual]

Returns the discount factor for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 635).

11.411.2.2 Rate forwardImpl (Time, bool *extrapolate* = false) const [protected, virtual]

Returns the instantaneous forward rate for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 635).

Reimplemented in **ZeroSpreadedTermStructure** (p. 682).

11.411.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 636).

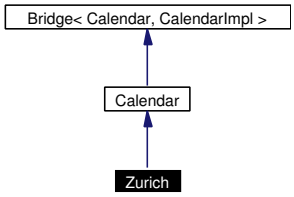
The documentation for this class was generated from the following file:

- ql/**termstructure.hpp**

11.412 Zurich Class Reference

`#include <zurich.hpp>`

Inheritance diagram for Zurich:



11.412.1 Detailed Description

Zurich calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Berchtoldstag, January 2nd
- Good Friday
- Easter Monday
- Ascension Day
- Whit Monday
- Labour Day, May 1st
- National Day, August 1st
- Christmas, December 25th
- St. Stephen's Day, December 26th

The documentation for this class was generated from the following file:

- ql/Calendars/**zurich.hpp**

Chapter 12

QuantLib File Documentation

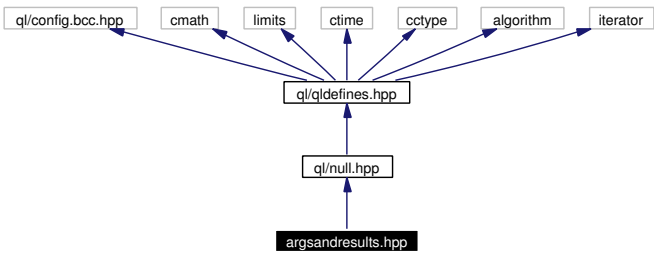
12.1 ql/argsandresults.hpp File Reference

12.1.1 Detailed Description

Base classes for generic arguments and results.

`#include <ql/null.hpp>`

Include dependency graph for argsandresults.hpp:



Namespaces

- namespace **QuantLib**

Defines

- #define **QL_MIN_VOLATILITY** 0.0
- #define **QL_MIN_DIVYIELD** 1.0e-7
- #define **QL_MAX_VOLATILITY** 4.0
- #define **QL_MAX_DIVYIELD** 4.0

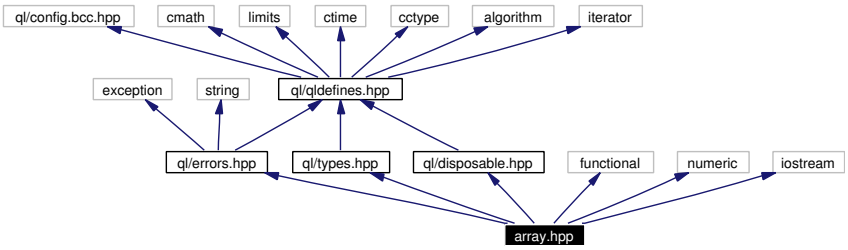
12.2 ql/array.hpp File Reference

12.2.1 Detailed Description

1-D array used in linear algebra.

```
#include <ql/types.hpp>
#include <ql/errors.hpp>
#include <ql/disposable.hpp>
#include <functional>
#include <numeric>
#include <iostream>
```

Include dependency graph for array.hpp:



Namespaces

- namespace **QuantLib**

12.3 ql/blackmodel.hpp File Reference

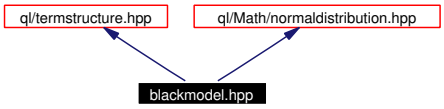
12.3.1 Detailed Description

Abstract class for Black-type models (market models).

`#include <ql/termstructure.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for blackmodel.hpp:



Namespaces

- namespace **QuantLib**

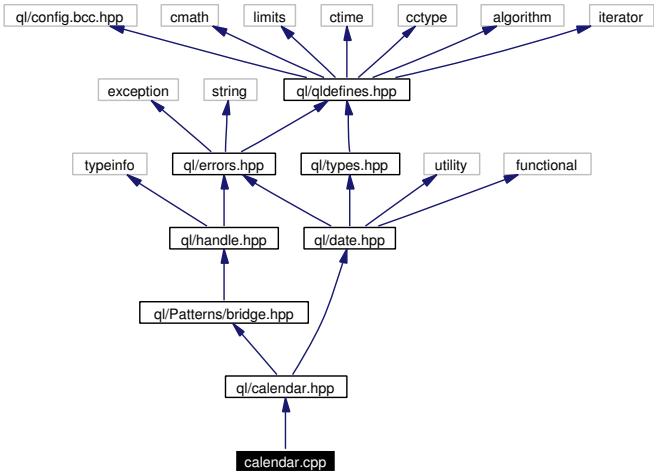
12.4 ql/calendar.cpp File Reference

12.4.1 Detailed Description

Abstract calendar class.

`#include <ql/calendar.hpp>`

Include dependency graph for calendar.cpp:



Namespaces

- namespace **QuantLib**

12.5 ql/calendar.hpp File Reference

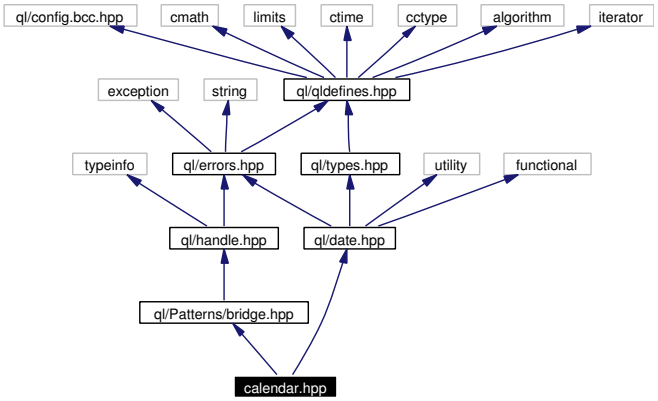
12.5.1 Detailed Description

calendar class

`#include <ql/date.hpp>`

`#include <ql/Patterns/bridge.hpp>`

Include dependency graph for calendar.hpp:



Namespaces

- namespace **QuantLib::Calendars**
- namespace **QuantLib**

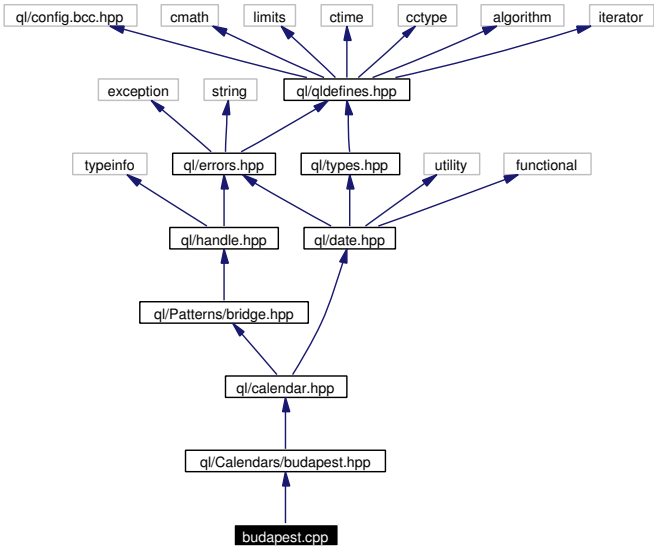
12.6 ql/Calendars/budapest.cpp File Reference

12.6.1 Detailed Description

Budapest calendar.

`#include <ql/Calendars/budapest.hpp>`

Include dependency graph for budapest.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

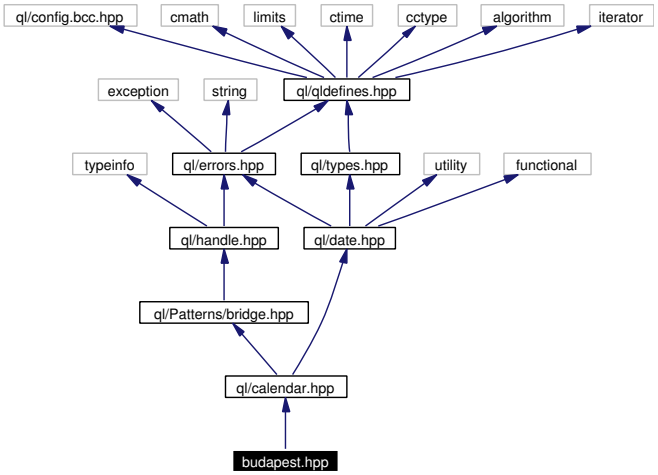
12.7 ql/Calendars/budapest.hpp File Reference

12.7.1 Detailed Description

Budapest calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for budapest.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

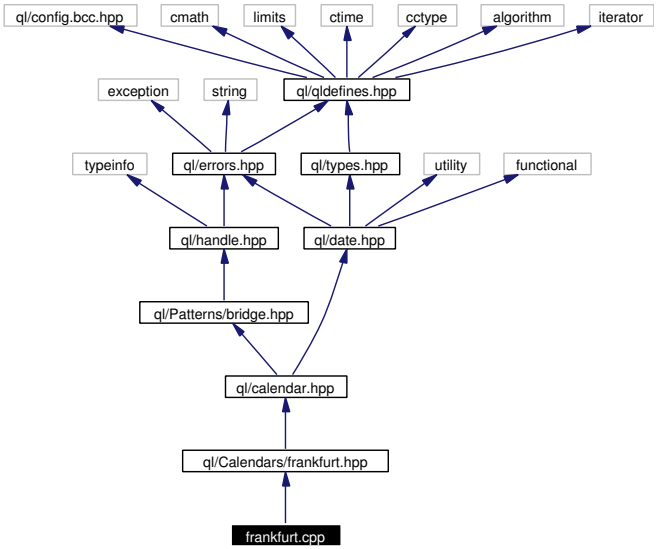
12.8 ql/Calendars/frankfurt.cpp File Reference

12.8.1 Detailed Description

Frankfurt calendar.

`#include <ql/Calendars/frankfurt.hpp>`

Include dependency graph for frankfurt.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

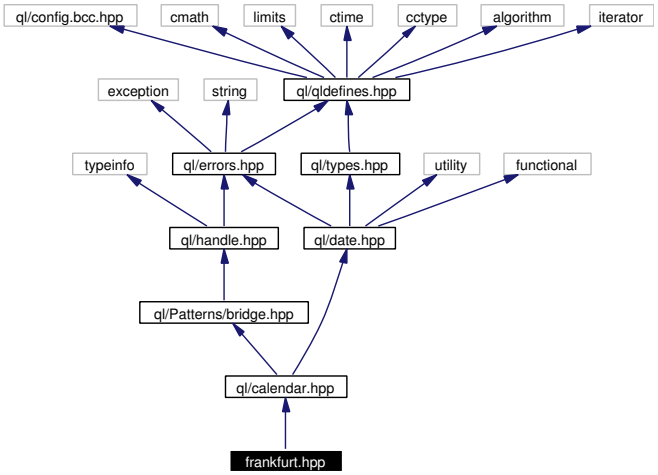
12.9 ql/Calendars/frankfurt.hpp File Reference

12.9.1 Detailed Description

Frankfurt calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for frankfurt.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

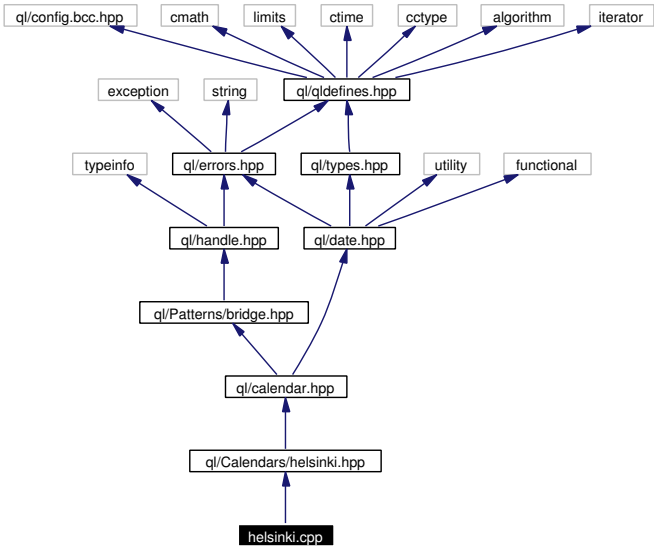
12.10 ql/Calendars/helsinki.cpp File Reference

12.10.1 Detailed Description

Helsinki calendar.

`#include <ql/Calendars/helsinki.hpp>`

Include dependency graph for helsinki.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

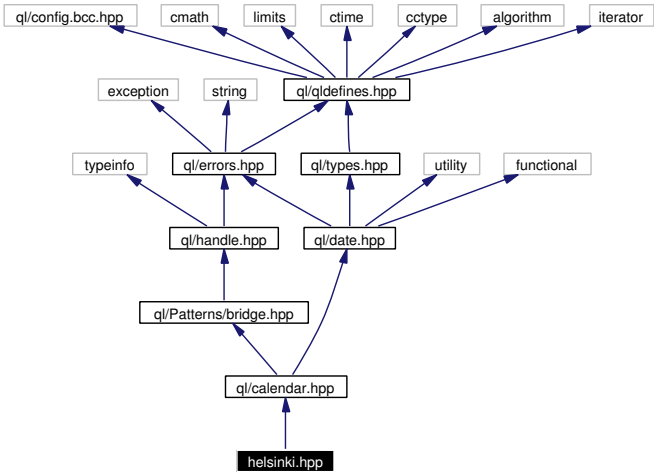
12.11 ql/Calendars/helsinki.hpp File Reference

12.11.1 Detailed Description

Helsinki calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for helsinki.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

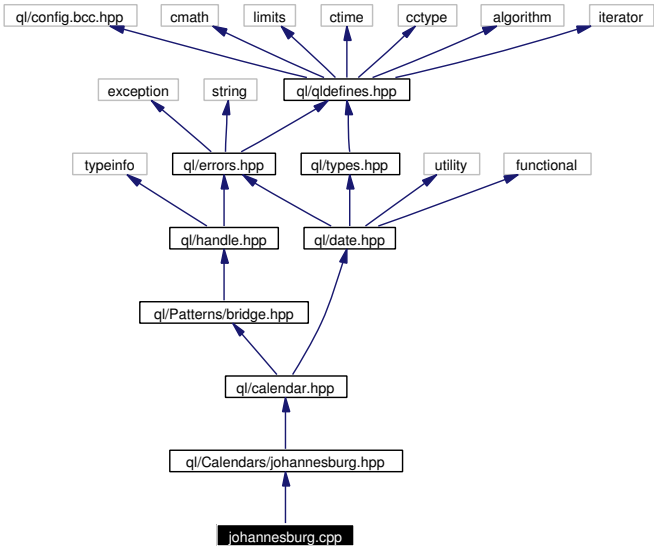
12.12 ql/Calendars/johannesburg.cpp File Reference

12.12.1 Detailed Description

Johannesburg calendar.

`#include <ql/Calendars/johannesburg.hpp>`

Include dependency graph for johannesburg.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

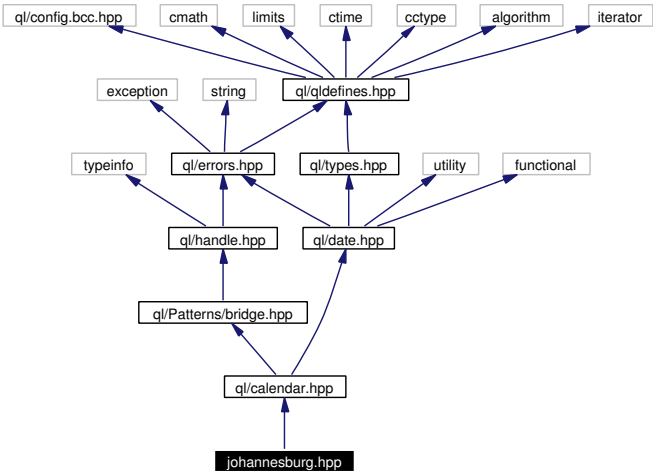
12.13 ql/Calendars/johannesburg.hpp File Reference

12.13.1 Detailed Description

Johannesburg calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for johannesburg.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

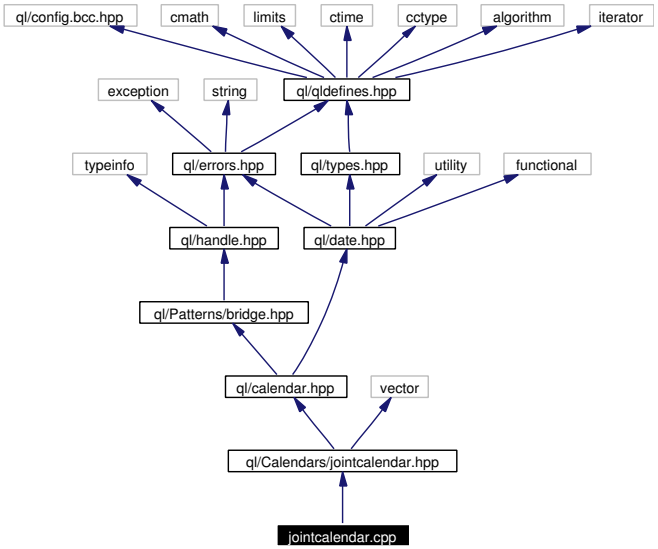
12.14 ql/Calendars/jointcalendar.cpp File Reference

12.14.1 Detailed Description

Joint calendar.

`#include <ql/Calendars/jointcalendar.hpp>`

Include dependency graph for jointcalendar.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

12.15 ql/Calendars/jointcalendar.hpp File Reference

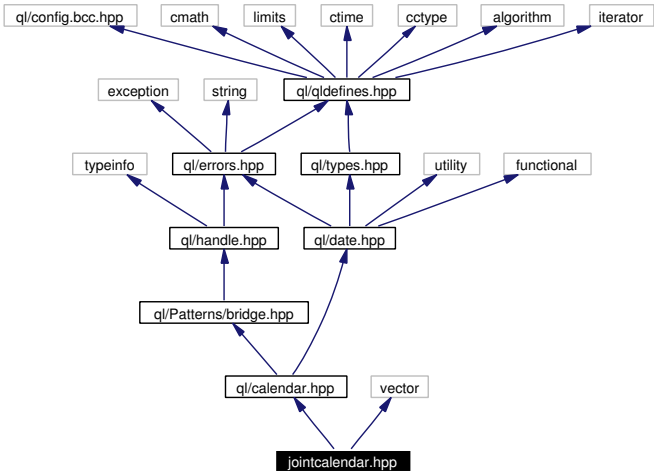
12.15.1 Detailed Description

Joint calendar.

`#include <ql/calendar.hpp>`

`#include <vector>`

Include dependency graph for jointcalendar.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

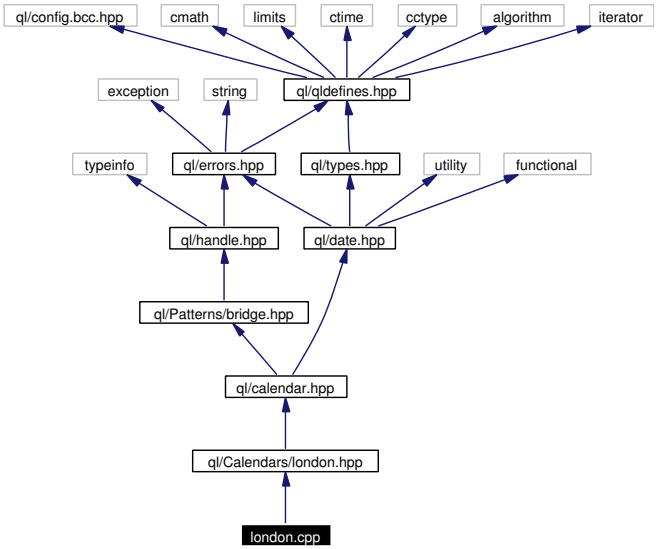
12.16 ql/Calendars/london.cpp File Reference

12.16.1 Detailed Description

London calendar.

`#include <ql/Calendars/london.hpp>`

Include dependency graph for london.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

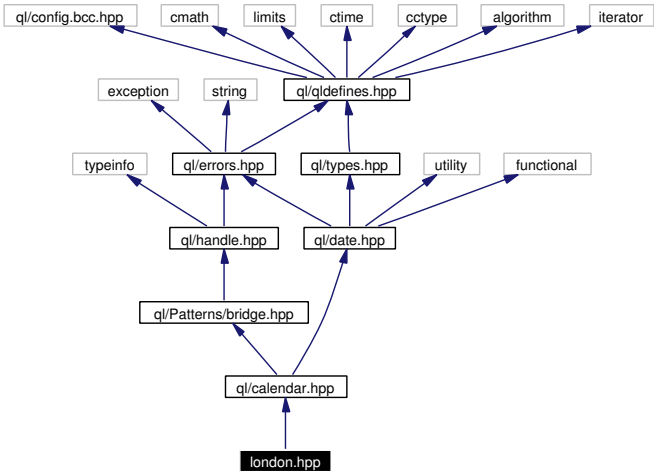
12.17 ql/Calendars/london.hpp File Reference

12.17.1 Detailed Description

London calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for london.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

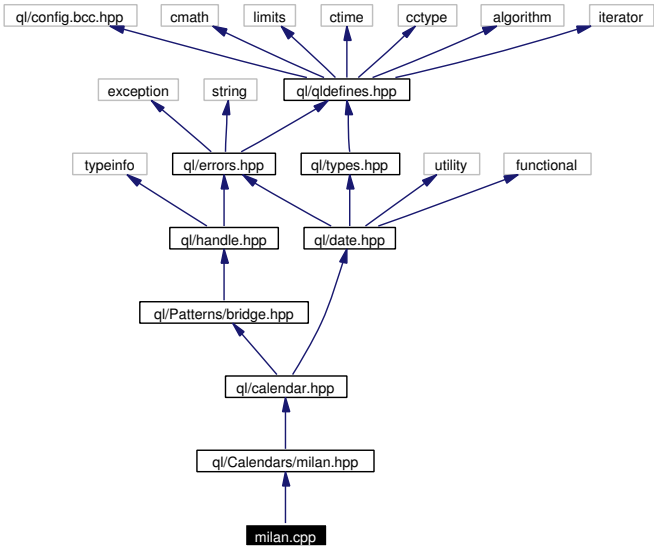
12.18 ql/Calendars/milan.cpp File Reference

12.18.1 Detailed Description

Milan calendar.

`#include <ql/Calendars/milan.hpp>`

Include dependency graph for milan.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

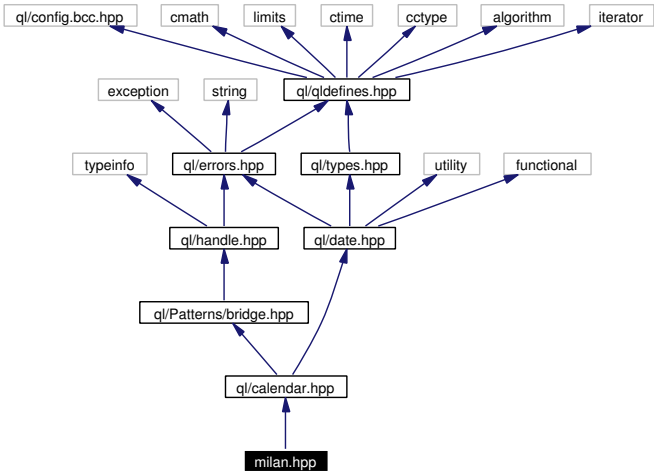
12.19 ql/Calendars/milan.hpp File Reference

12.19.1 Detailed Description

Milan calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for milan.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

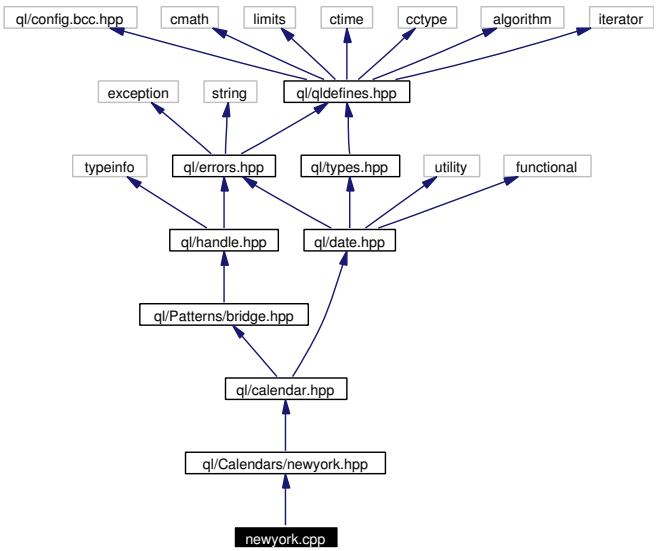
12.20 ql/Calendars/newyork.cpp File Reference

12.20.1 Detailed Description

New York calendar.

`#include <ql/Calendars/newyork.hpp>`

Include dependency graph for newyork.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

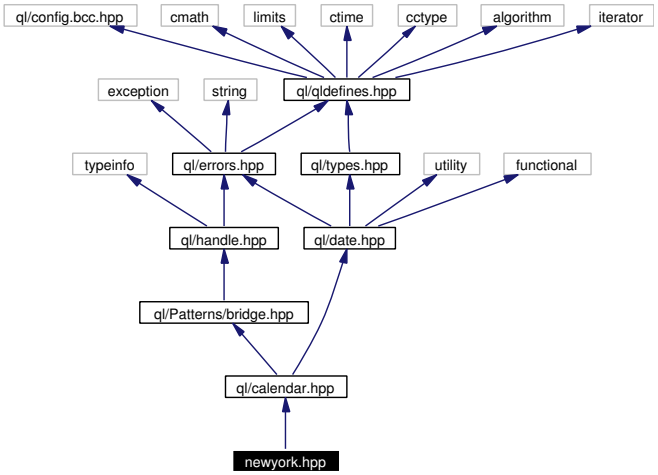
12.21 ql/Calendars/newyork.hpp File Reference

12.21.1 Detailed Description

New York calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for newyork.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

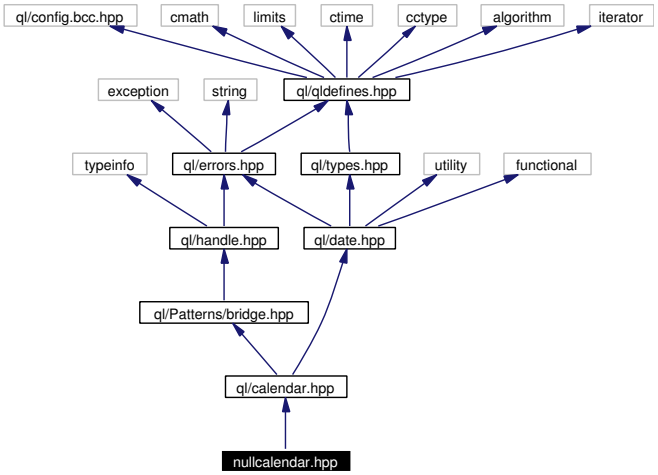
12.22 ql/Calendars/nullcalendar.hpp File Reference

12.22.1 Detailed Description

Calendar for reproducing theoretical calculations.

`#include <ql/calendar.hpp>`

Include dependency graph for nullcalendar.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

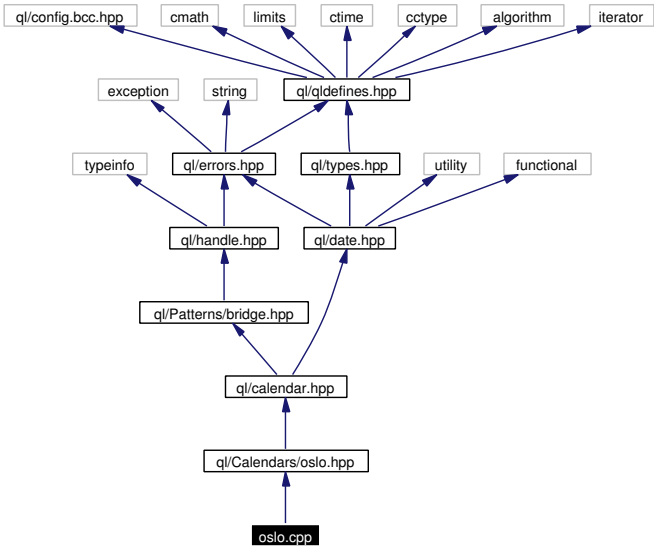
12.23 ql/Calendars/oslo.cpp File Reference

12.23.1 Detailed Description

Oslo calendar.

`#include <ql/Calendars/oslo.hpp>`

Include dependency graph for oslo.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

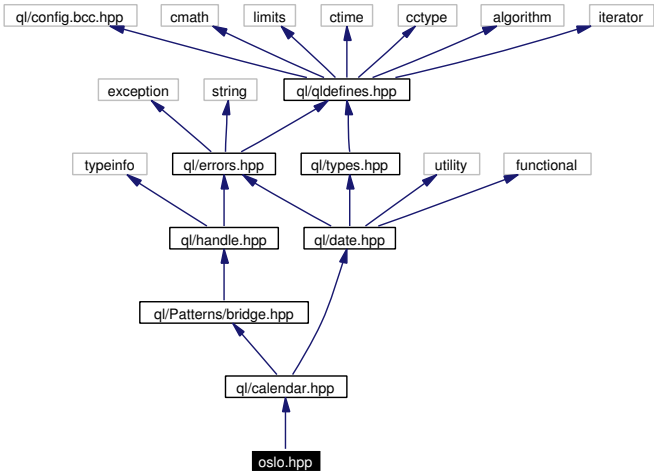
12.24 ql/Calendars/oslo.hpp File Reference

12.24.1 Detailed Description

Oslo calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for oslo.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

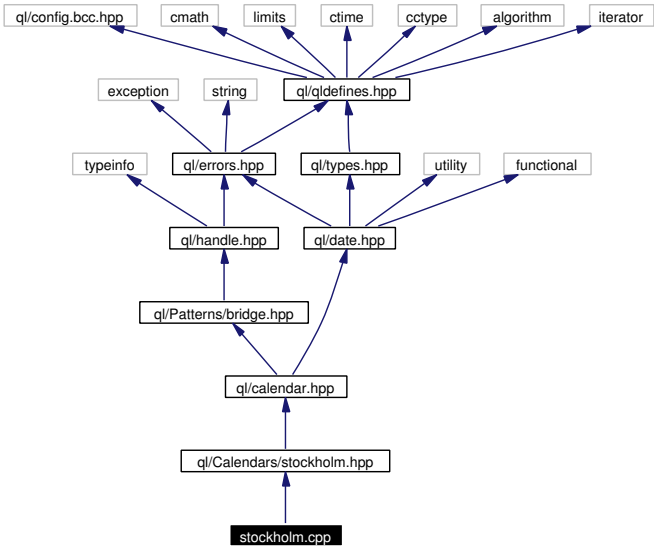
12.25 ql/Calendars/stockholm.cpp File Reference

12.25.1 Detailed Description

Stockholm calendar.

`#include <ql/Calendars/stockholm.hpp>`

Include dependency graph for stockholm.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

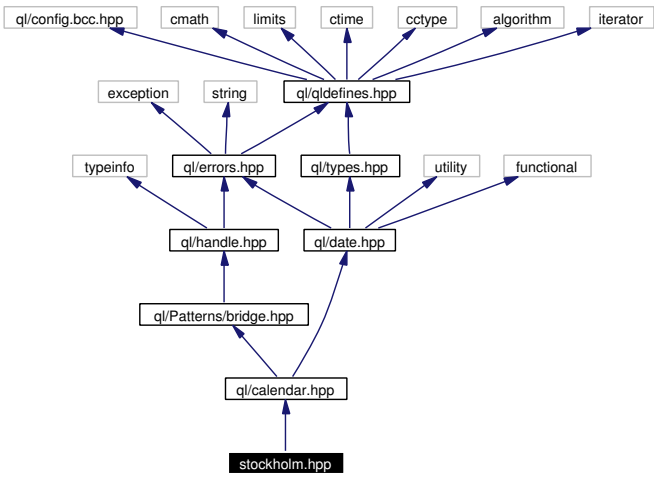
12.26 ql/Calendars/stockholm.hpp File Reference

12.26.1 Detailed Description

Stockholm calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for stockholm.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

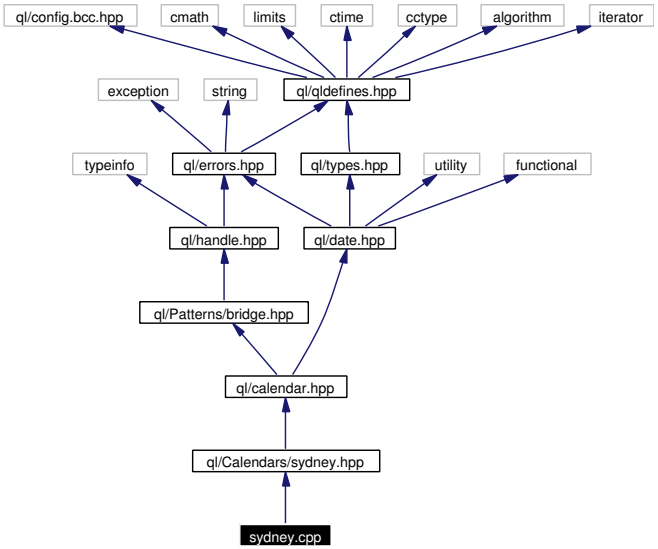
12.27 ql/Calendars/sydney.cpp File Reference

12.27.1 Detailed Description

Sydney calendar.

`#include <ql/Calendars/sydney.hpp>`

Include dependency graph for sydney.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

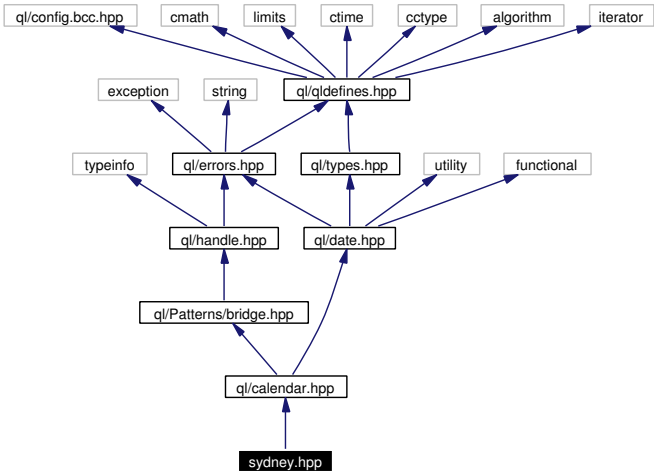
12.28 ql/Calendars/sydney.hpp File Reference

12.28.1 Detailed Description

Sydney calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for sydney.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

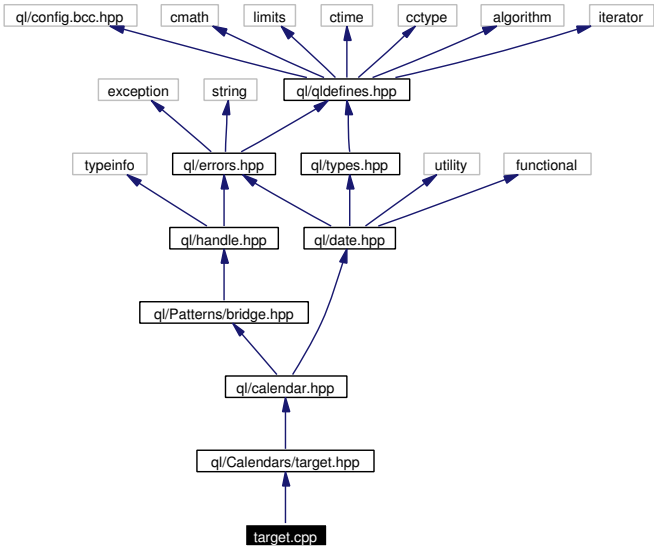
12.29 ql/Calendars/target.cpp File Reference

12.29.1 Detailed Description

TARGET calendar.

`#include <ql/Calendars/target.hpp>`

Include dependency graph for target.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

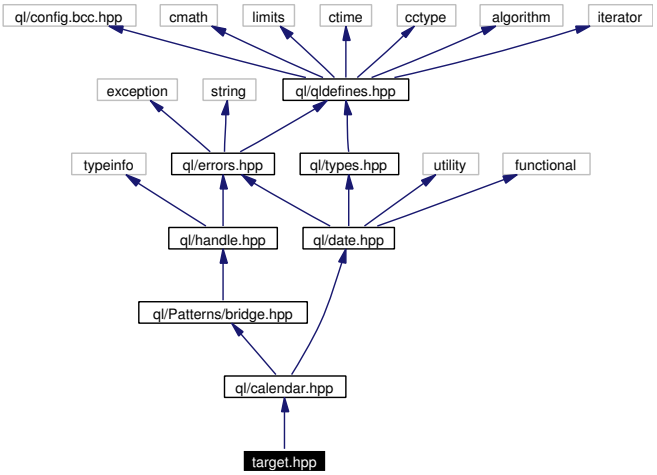
12.30 ql/Calendars/target.hpp File Reference

12.30.1 Detailed Description

TARGET calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for target.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

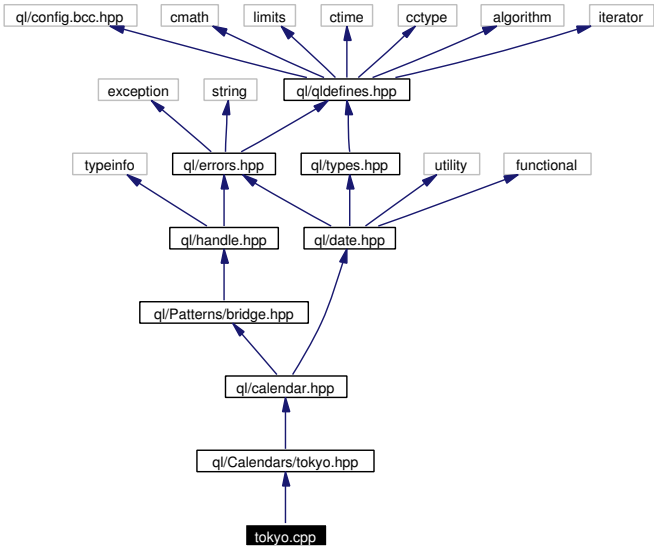
12.31 ql/Calendars/tokyo.cpp File Reference

12.31.1 Detailed Description

Tokyo calendar.

`#include <ql/Calendars/tokyo.hpp>`

Include dependency graph for tokyo.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

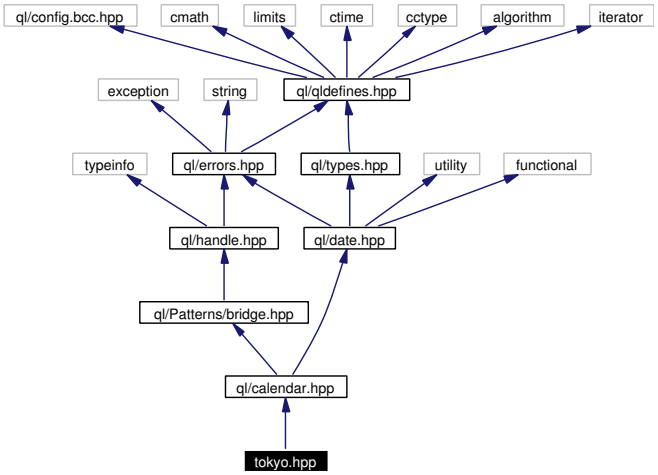
12.32 ql/Calendars/tokyo.hpp File Reference

12.32.1 Detailed Description

Tokyo calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for tokyo.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

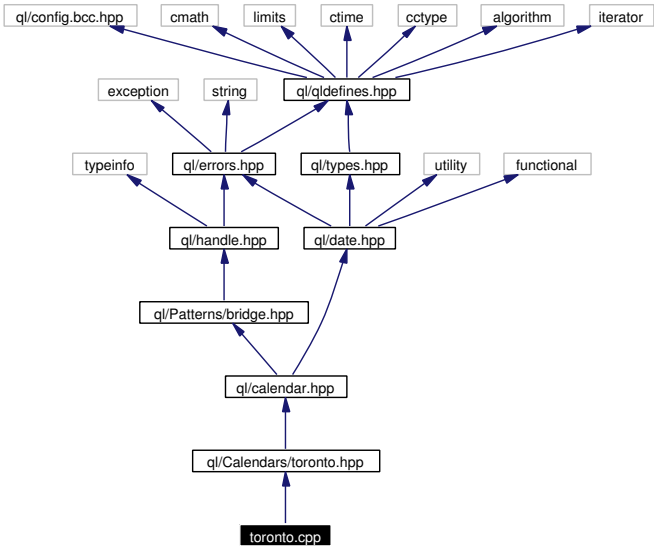
12.33 ql/Calendars/toronto.cpp File Reference

12.33.1 Detailed Description

Toronto calendar.

`#include <ql/Calendars/toronto.hpp>`

Include dependency graph for toronto.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

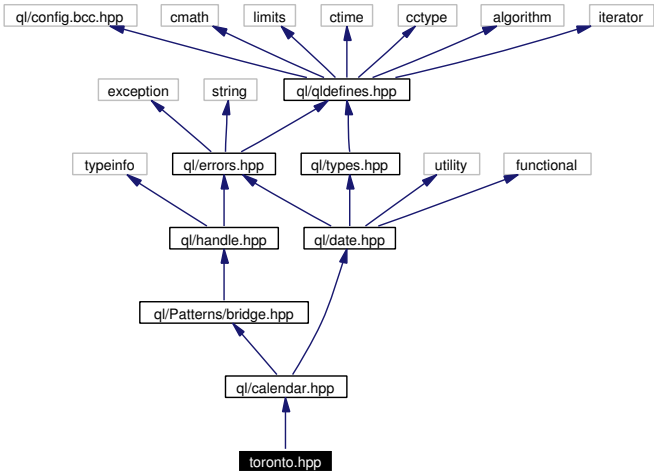
12.34 ql/Calendars/toronto.hpp File Reference

12.34.1 Detailed Description

Toronto calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for toronto.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

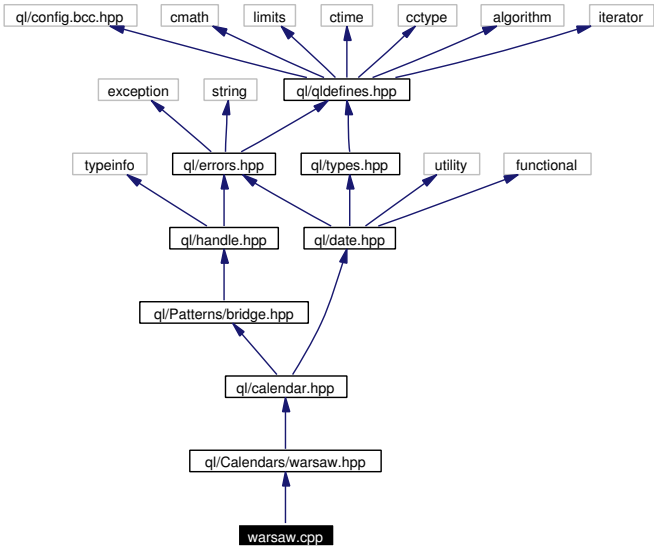
12.35 ql/Calendars/warsaw.cpp File Reference

12.35.1 Detailed Description

Warsaw calendar.

`#include <ql/Calendars/warsaw.hpp>`

Include dependency graph for warsaw.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

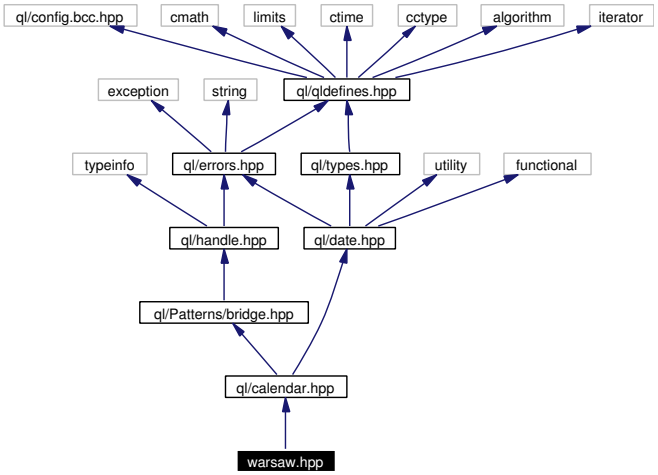
12.36 ql/Calendars/warsaw.hpp File Reference

12.36.1 Detailed Description

Warsaw calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for warsaw.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

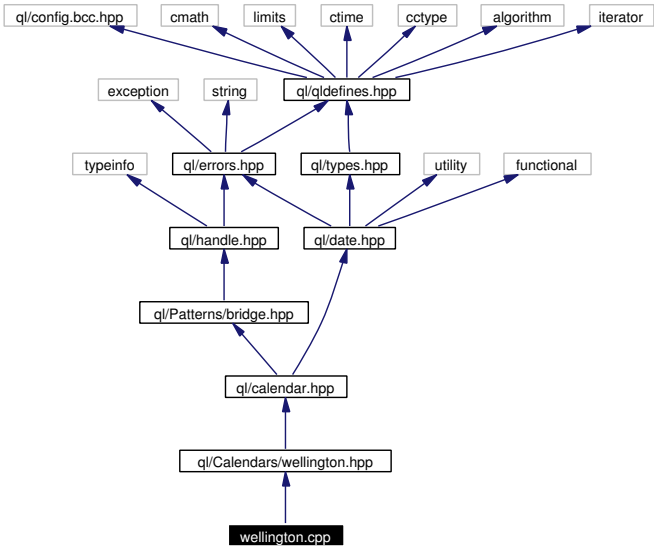
12.37 ql/Calendars/wellington.cpp File Reference

12.37.1 Detailed Description

Wellington calendar.

`#include <ql/Calendars/wellington.hpp>`

Include dependency graph for wellington.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

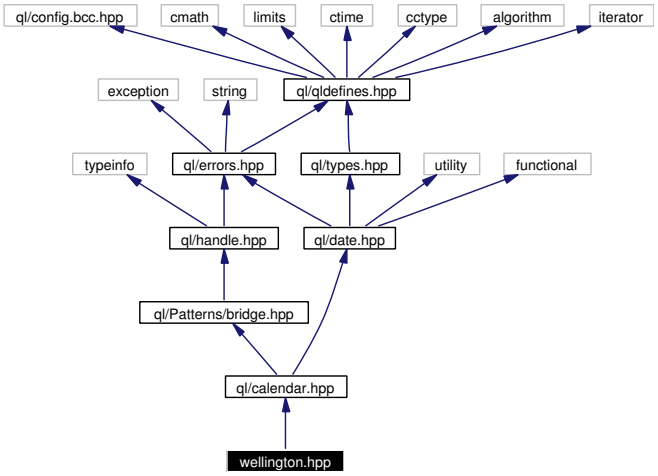
12.38 ql/Calendars/wellington.hpp File Reference

12.38.1 Detailed Description

Wellington calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for wellington.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

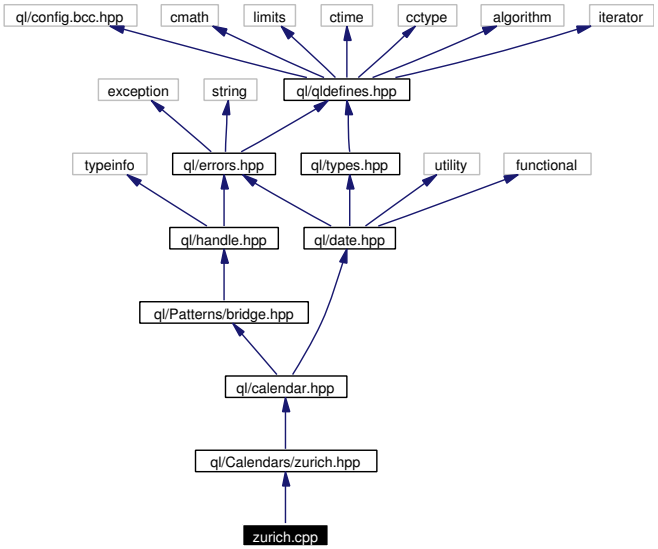
12.39 ql/Calendars/zurich.cpp File Reference

12.39.1 Detailed Description

Zurich calendar.

`#include <ql/Calendars/zurich.hpp>`

Include dependency graph for zurich.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

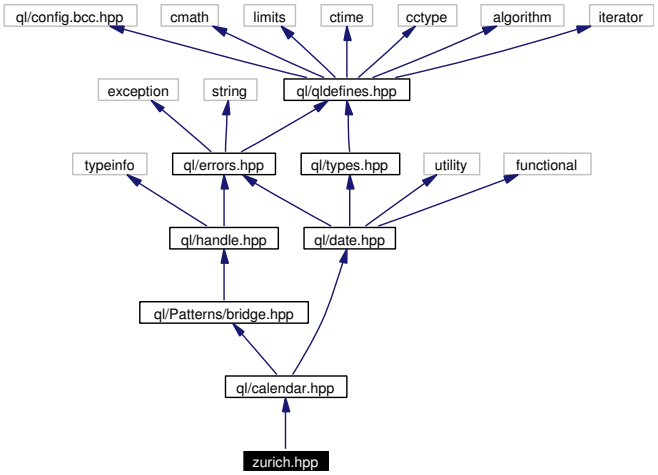
12.40 ql/Calendars/zurich.hpp File Reference

12.40.1 Detailed Description

Zurich calendar.

`#include <ql/calendar.hpp>`

Include dependency graph for zurich.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Calendars**

12.41 ql/capvolstructures.hpp File Reference

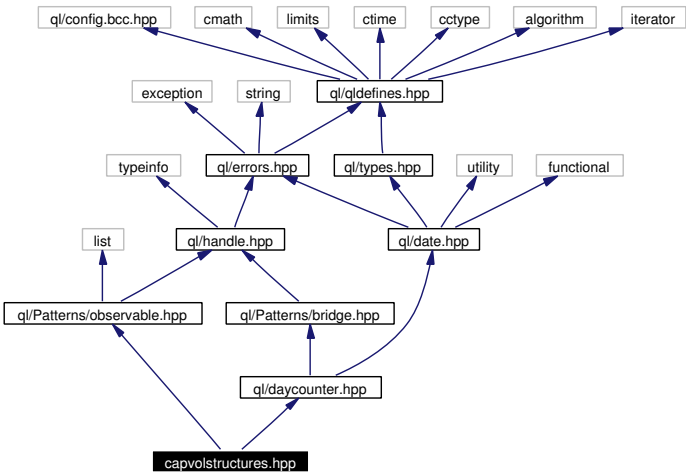
12.41.1 Detailed Description

Cap/Floor volatility structures.

`#include <ql/daycounter.hpp>`

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for capvolstructures.hpp:



Namespaces

- namespace **QuantLib**

12.42 ql/cashflow.hpp File Reference

12.42.1 Detailed Description

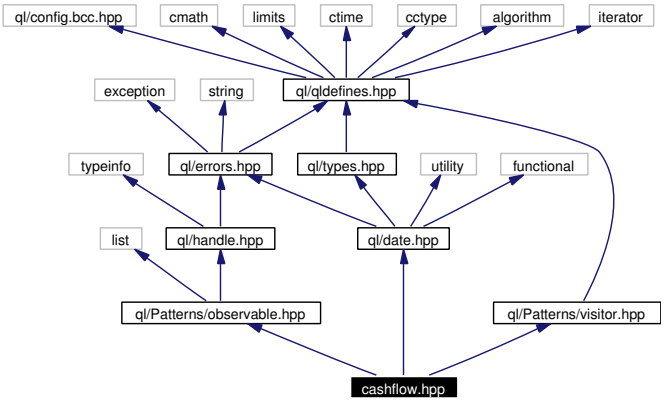
Base class for cash flows.

`#include <ql/date.hpp>`

`#include <ql/Patterns/observable.hpp>`

`#include <ql/Patterns/visitor.hpp>`

Include dependency graph for cashflow.hpp:



Namespaces

- namespace **QuantLib::CashFlows**
- namespace **QuantLib**

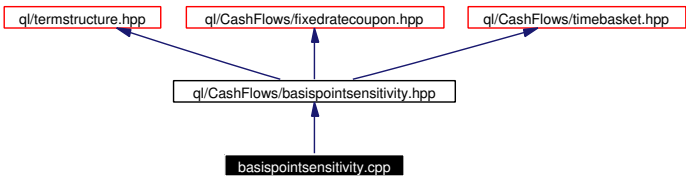
12.43 ql/CashFlows/basispointsensitivity.cpp File Reference

12.43.1 Detailed Description

basis point sensitivity calculator

`#include <ql/CashFlows/basispointsensitivity.hpp>`

Include dependency graph for basispointsensitivity.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.44 ql/CashFlows/basispointsensitivity.hpp File Reference

12.44.1 Detailed Description

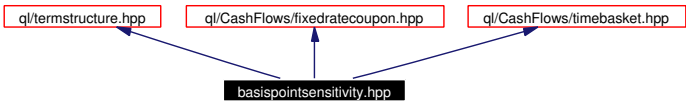
basis point sensitivity calculator

`#include <ql/termstructure.hpp>`

`#include <ql/CashFlows/fixedratecoupon.hpp>`

`#include <ql/CashFlows/timebasket.hpp>`

Include dependency graph for basispointsensitivity.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.45 ql/CashFlows/cashflowvectors.cpp File Reference

12.45.1 Detailed Description

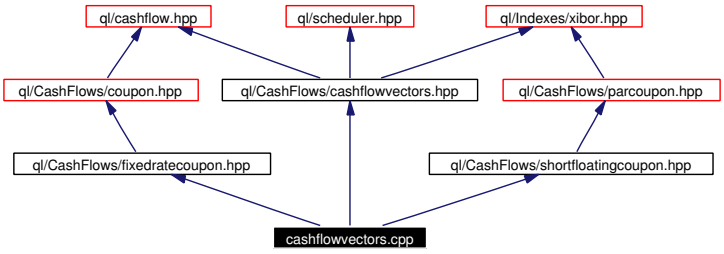
Cash flow vector builders.

`#include <ql/CashFlows/cashflowvectors.hpp>`

`#include <ql/CashFlows/fixedratecoupon.hpp>`

`#include <ql/CashFlows/shortfloatingcoupon.hpp>`

Include dependency graph for cashflowvectors.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.46 ql/CashFlows/cashflowvectors.hpp File Reference

12.46.1 Detailed Description

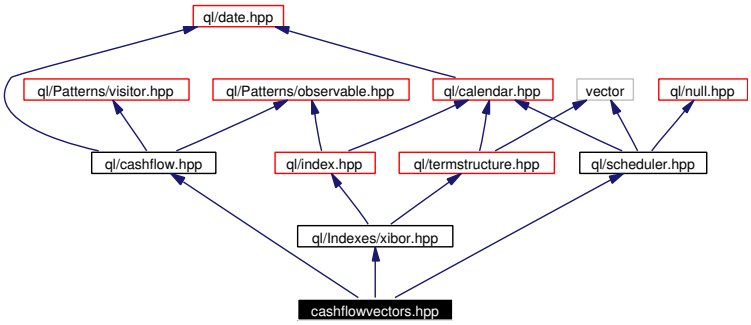
Cash flow vector builders.

`#include <ql/cashflow.hpp>`

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/scheduler.hpp>`

Include dependency graph for cashflowvectors.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.47 ql/CashFlows/coupon.hpp File Reference

12.47.1 Detailed Description

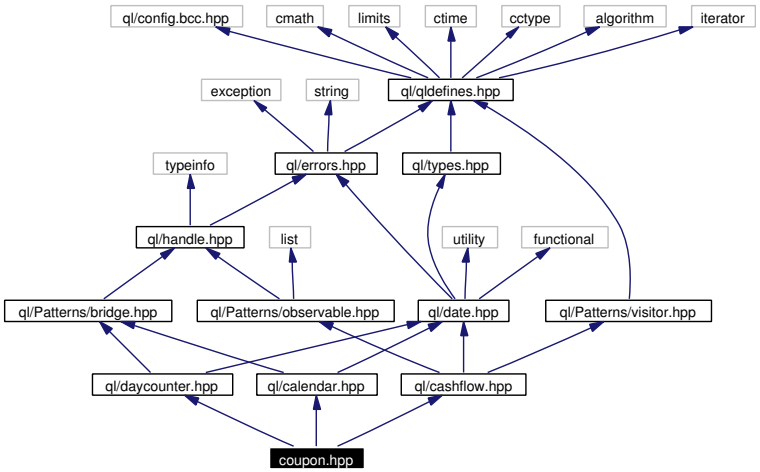
Coupon accruing over a fixed period.

`#include <ql/cashflow.hpp>`

`#include <ql/calendar.hpp>`

`#include <ql/daycounter.hpp>`

Include dependency graph for coupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

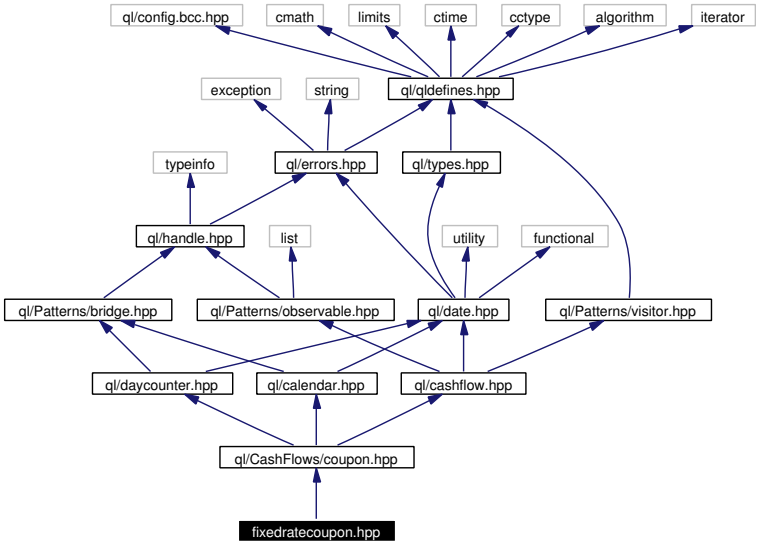
12.48 ql/CashFlows/fixedratecoupon.hpp File Reference

12.48.1 Detailed Description

Coupon paying a fixed annual rate.

`#include <ql/CashFlows/coupon.hpp>`

Include dependency graph for fixedratecoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

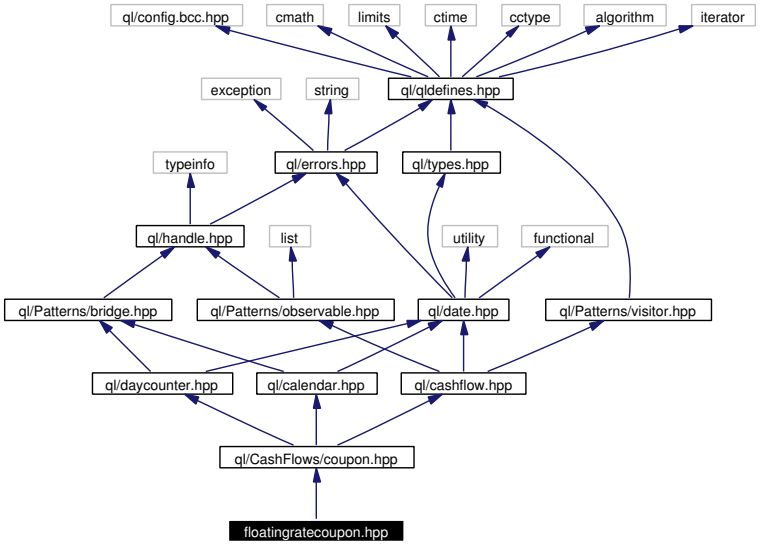
12.49 ql/CashFlows/floatingratecoupon.hpp File Reference

12.49.1 Detailed Description

Coupon at par on a term structure.

`#include <ql/CashFlows/coupon.hpp>`

Include dependency graph for floatingratecoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

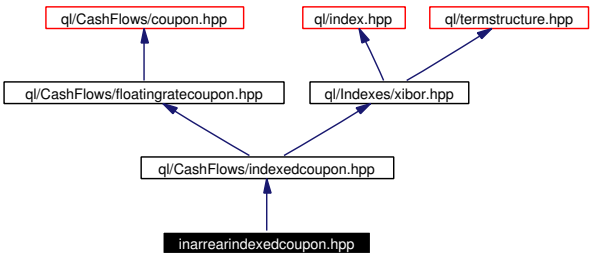
12.50 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

12.50.1 Detailed Description

in arrear indexed coupon

`#include <ql/CashFlows/indexedcoupon.hpp>`

Include dependency graph for inarrearindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.51 ql/CashFlows/indexcashflowvectors.hpp File Reference

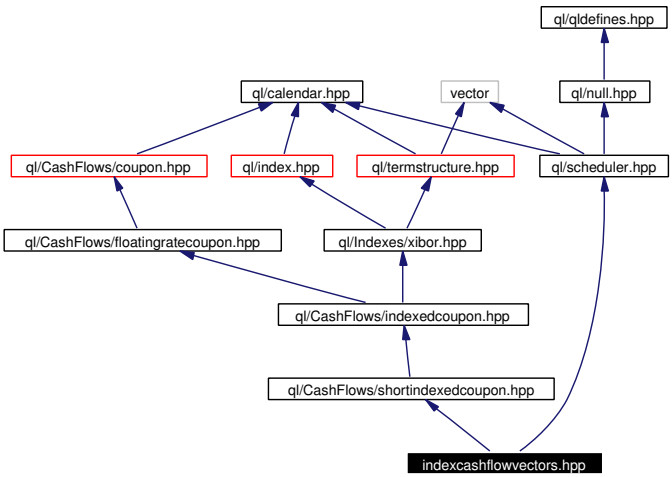
12.51.1 Detailed Description

Index Cash flow vector builders.

`#include <ql/CashFlows/shortindexedcoupon.hpp>`

`#include <ql/scheduler.hpp>`

Include dependency graph for indexcashflowvectors.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.52 ql/CashFlows/indexedcoupon.hpp File Reference

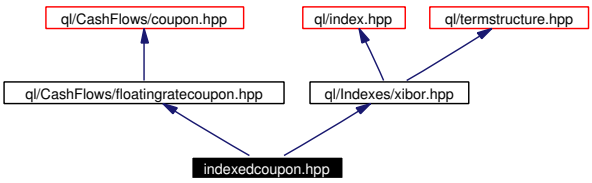
12.52.1 Detailed Description

indexed coupon

`#include <ql/CashFlows/floatingratecoupon.hpp>`

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for indexedcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.53 ql/CashFlows/parcoupon.cpp File Reference

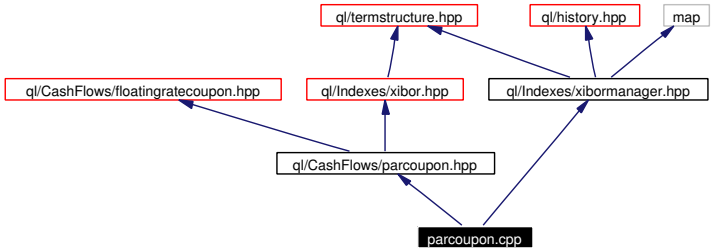
12.53.1 Detailed Description

Coupon at par on a term structure.

`#include <ql/CashFlows/parcoupon.hpp>`

`#include <ql/Indexes/xibormanager.hpp>`

Include dependency graph for parcoupon.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.54 ql/CashFlows/parcoupon.hpp File Reference

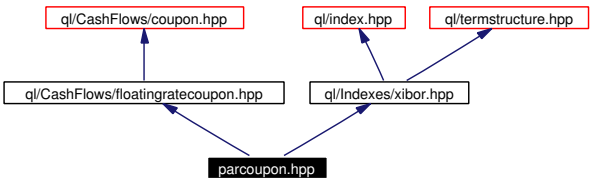
12.54.1 Detailed Description

Coupon at par on a term structure.

`#include <ql/CashFlows/floatingratecoupon.hpp>`

`#include <ql/Indexes/xibor.hpp>`

Include dependency graph for parcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

12.55 ql/CashFlows/shortfloatingcoupon.cpp File Reference

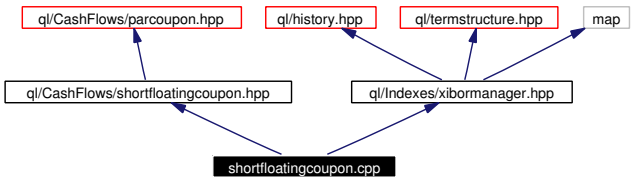
12.55.1 Detailed Description

Short coupon at par on a term structure.

`#include <ql/CashFlows/shortfloatingcoupon.hpp>`

`#include <ql/Indexes/xibormanager.hpp>`

Include dependency graph for shortfloatingcoupon.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

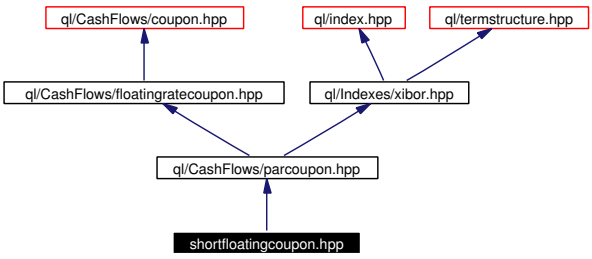
12.56 ql/CashFlows/shortfloatingcoupon.hpp File Reference

12.56.1 Detailed Description

Short (or long) coupon at par on a term structure.

`#include <ql/CashFlows/parcoupon.hpp>`

Include dependency graph for shortfloatingcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

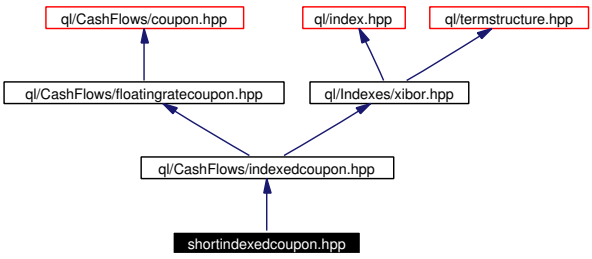
12.57 ql/CashFlows/shortindexedcoupon.hpp File Reference

12.57.1 Detailed Description

Short (or long) indexed coupon.

`#include <ql/CashFlows/indexedcoupon.hpp>`

Include dependency graph for shortindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

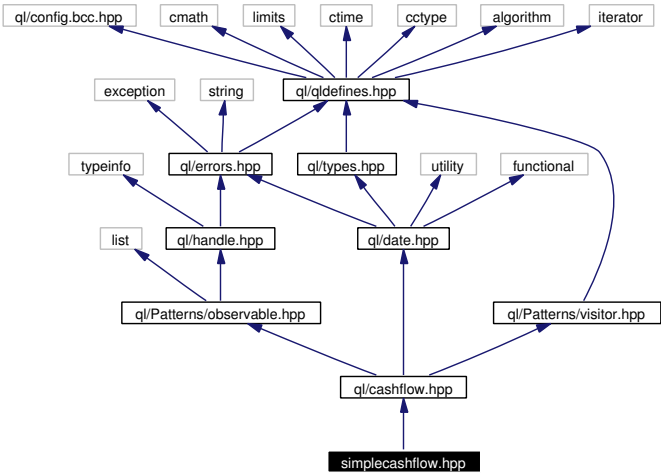
12.58 ql/CashFlows/simplecashflow.hpp File Reference

12.58.1 Detailed Description

Predetermined cash flow.

`#include <ql/cashflow.hpp>`

Include dependency graph for simplecashflow.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

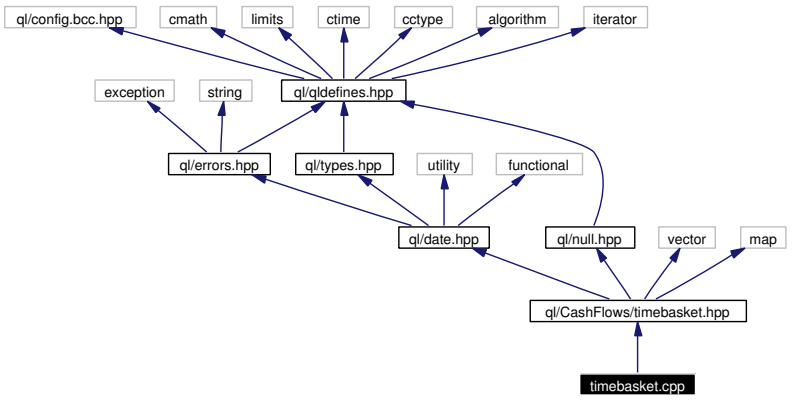
12.59 ql/CashFlows/timebasket.cpp File Reference

12.59.1 Detailed Description

Distribution over a number of dates

`#include <ql/CashFlows/timebasket.hpp>`

Include dependency graph for timebasket.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

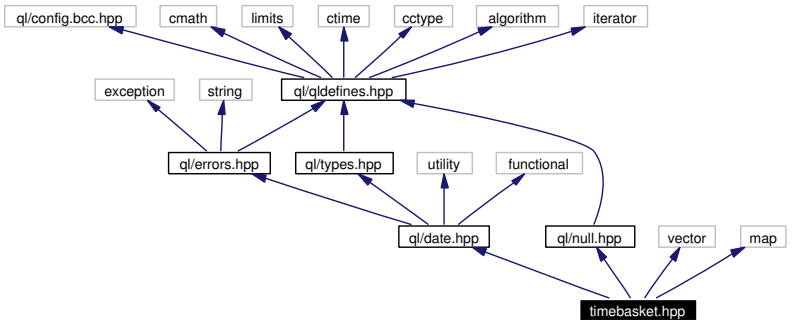
12.60 ql/CashFlows/timebasket.hpp File Reference

12.60.1 Detailed Description

Distribution over a number of dates

```
#include <ql/date.hpp>
#include <ql/null.hpp>
#include <vector>
#include <map>
```

Include dependency graph for timebasket.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

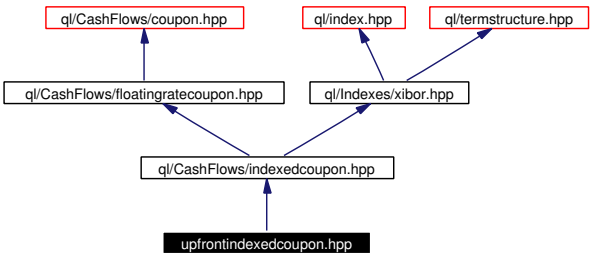
12.61 ql/CashFlows/upfrontindexedcoupon.hpp File Reference

12.61.1 Detailed Description

Up front indexed coupon.

`#include <ql/CashFlows/indexedcoupon.hpp>`

Include dependency graph for upfrontindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::CashFlows**

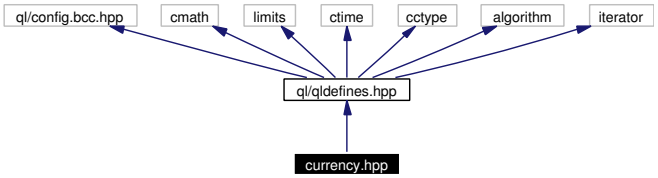
12.62 ql/currency.hpp File Reference

12.62.1 Detailed Description

Known currencies.

`#include <ql/qldefines.hpp>`

Include dependency graph for currency.hpp:



Namespaces

- namespace **QuantLib**

12.63 ql/dataformatters.cpp File Reference

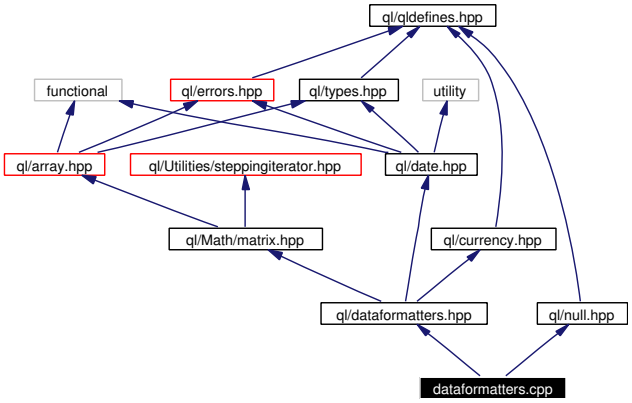
12.63.1 Detailed Description

classes used to format data for output

`#include <ql/dataformatters.hpp>`

`#include <ql/null.hpp>`

Include dependency graph for dataformatters.cpp:



Namespaces

- namespace **QuantLib**

12.64 ql/dataformatters.hpp File Reference

12.64.1 Detailed Description

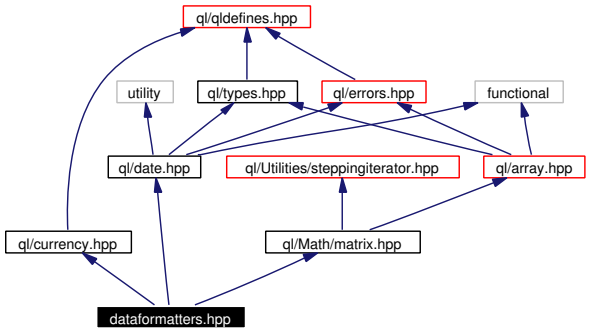
Classes used to format data for output.

`#include <ql/date.hpp>`

`#include <ql/currency.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for dataformatters.hpp:



Namespaces

- namespace **QuantLib**

12.65 ql/dataparsers.cpp File Reference

12.65.1 Detailed Description

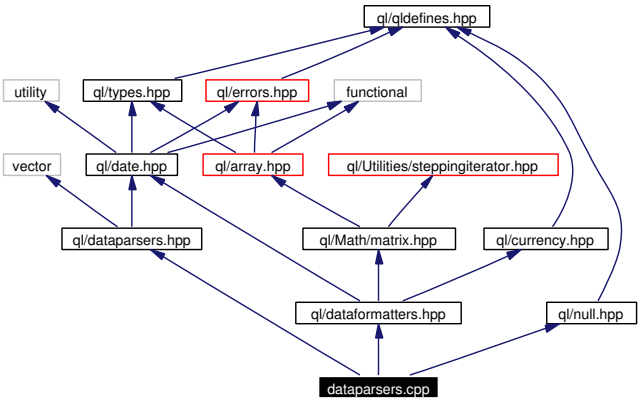
classes used to parse data for input

`#include <ql/dataparsers.hpp>`

`#include <ql/dataformatters.hpp>`

`#include <ql/null.hpp>`

Include dependency graph for dataparsers.cpp:



Namespaces

- namespace **QuantLib**

12.66 ql/dataparsers.hpp File Reference

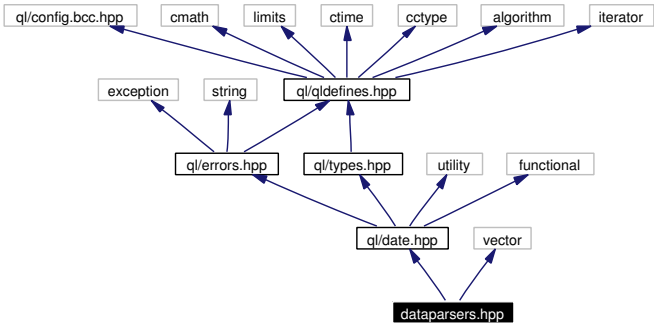
12.66.1 Detailed Description

Classes used to parse data for input.

`#include <ql/date.hpp>`

`#include <vector>`

Include dependency graph for dataparsers.hpp:



Namespaces

- namespace **QuantLib**

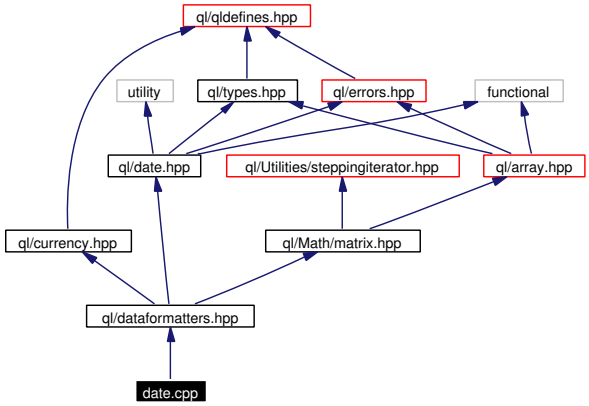
12.67 ql/date.cpp File Reference

12.67.1 Detailed Description

date- and time-related classes, typedefs and enumerations

`#include <ql/dataformatters.hpp>`

Include dependency graph for date.cpp:



Namespaces

- namespace **QuantLib**

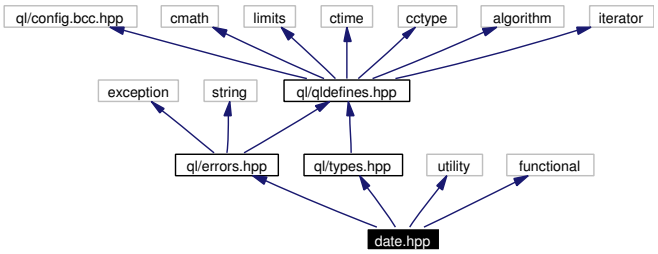
12.68 ql/date.hpp File Reference

12.68.1 Detailed Description

date- and time-related classes, typedefs and enumerations

```
#include <ql/errors.hpp>
#include <ql/types.hpp>
#include <utility>
#include <functional>
```

Include dependency graph for date.hpp:



Namespaces

- namespace **QuantLib**

12.69 ql/daycounter.hpp File Reference

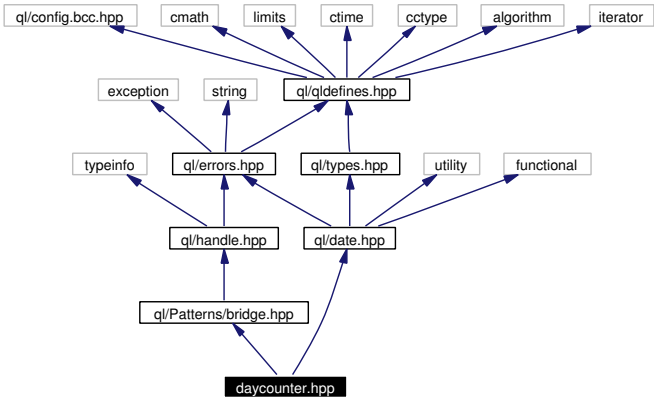
12.69.1 Detailed Description

day counter class

`#include <ql/date.hpp>`

`#include <ql/Patterns/bridge.hpp>`

Include dependency graph for daycounter.hpp:



Namespaces

- namespace **QuantLib::DayCounters**
- namespace **QuantLib**

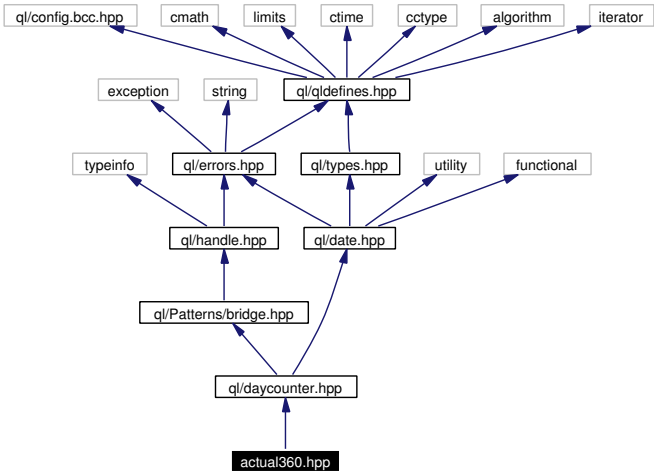
12.70 ql/DayCounters/actual360.hpp File Reference

12.70.1 Detailed Description

act/360 day counter

`#include <ql/daycounter.hpp>`

Include dependency graph for actual360.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

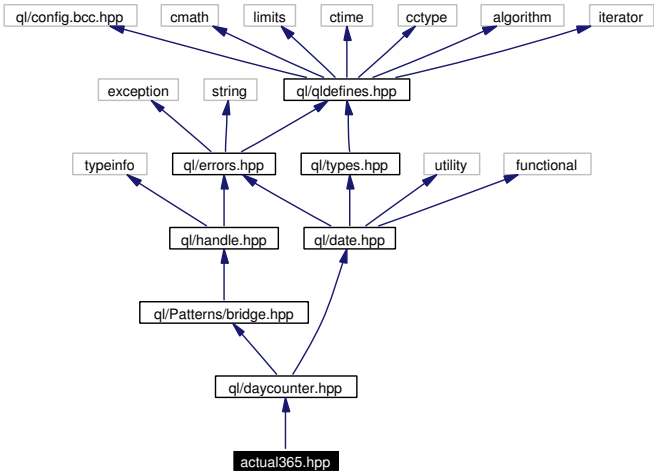
12.71 ql/DayCounters/actual365.hpp File Reference

12.71.1 Detailed Description

act/365 day counter

`#include <ql/daycounter.hpp>`

Include dependency graph for actual365.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

12.72 ql/DayCounters/actualactual.cpp File Reference

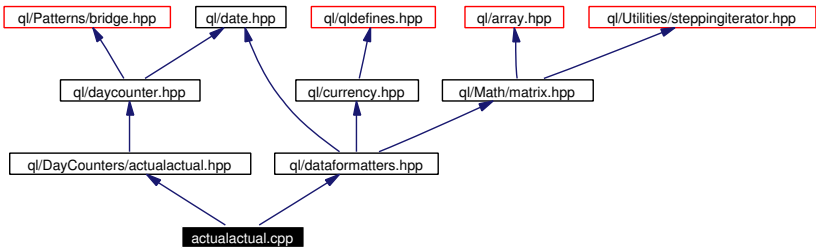
12.72.1 Detailed Description

act/act day counters

`#include <ql/DayCounters/actualactual.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for actualactual.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

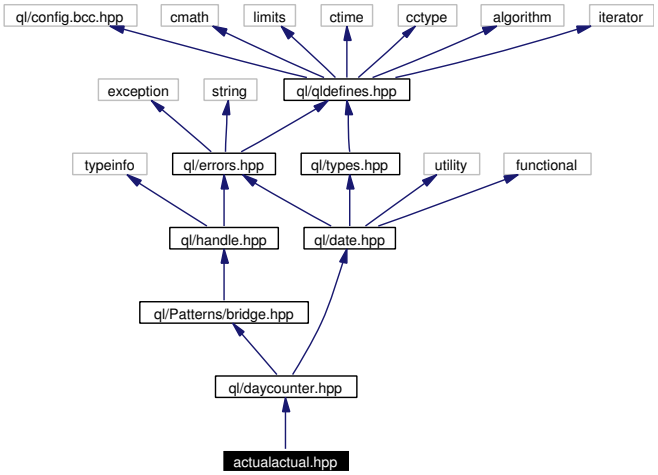
12.73 ql/DayCounters/actualactual.hpp File Reference

12.73.1 Detailed Description

act/act day counters

`#include <ql/daycounter.hpp>`

Include dependency graph for actualactual.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

12.74 ql/DayCounters/simpledaycounter.cpp File Reference

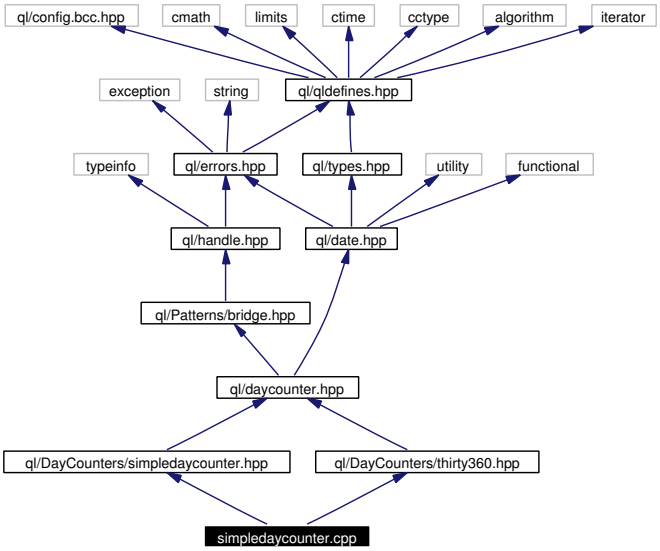
12.74.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

`#include <ql/DayCounters/simpledaycounter.hpp>`

`#include <ql/DayCounters/thirty360.hpp>`

Include dependency graph for simpledaycounter.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

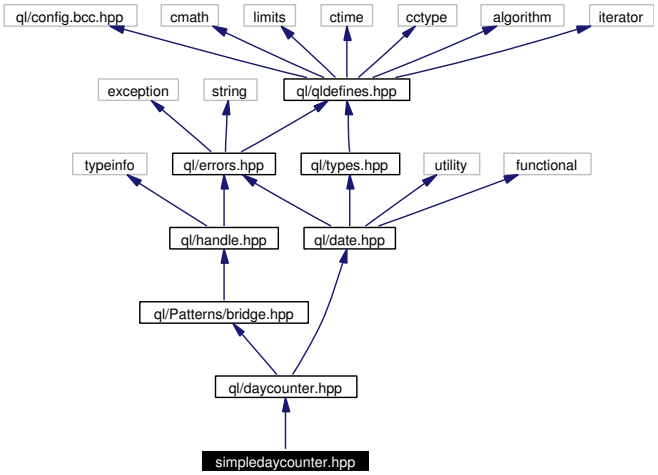
12.75 ql/DayCounters/simpledaycounter.hpp File Reference

12.75.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

`#include <ql/daycounter.hpp>`

Include dependency graph for simpledaycounter.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

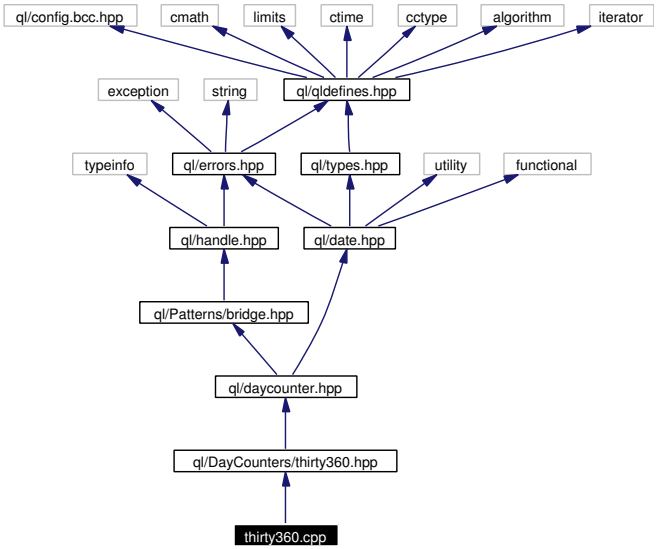
12.76 ql/DayCounters/thirty360.cpp File Reference

12.76.1 Detailed Description

30/360 day counters

`#include <ql/DayCounters/thirty360.hpp>`

Include dependency graph for thirty360.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

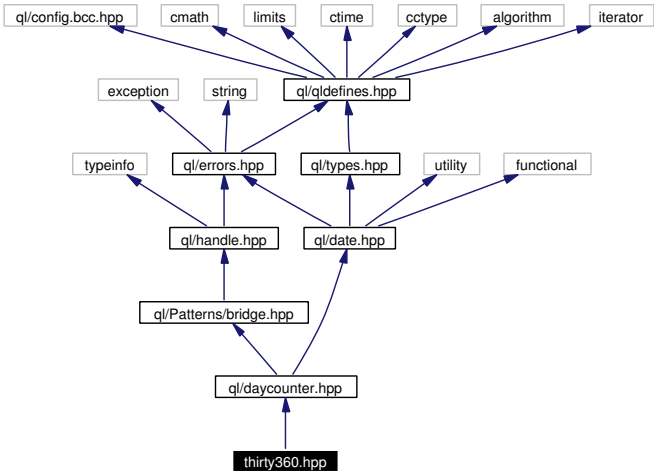
12.77 ql/DayCounters/thirty360.hpp File Reference

12.77.1 Detailed Description

30/360 day counters

`#include <ql/daycounter.hpp>`

Include dependency graph for thirty360.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::DayCounters**

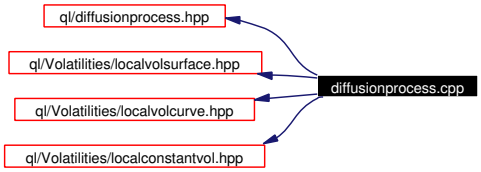
12.78 ql/diffusionprocess.cpp File Reference

12.78.1 Detailed Description

Diffusion process.

```
#include <ql/diffusionprocess.hpp>
#include <ql/Volatilities/localvolsurface.hpp>
#include <ql/Volatilities/localvolcurve.hpp>
#include <ql/Volatilities/localconstantvol.hpp>
```

Include dependency graph for diffusionprocess.cpp:



Namespaces

- namespace **QuantLib**

12.79 ql/diffusionprocess.hpp File Reference

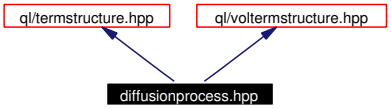
12.79.1 Detailed Description

Diffusion process.

`#include <ql/termstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for diffusionprocess.hpp:



Namespaces

- namespace **QuantLib**

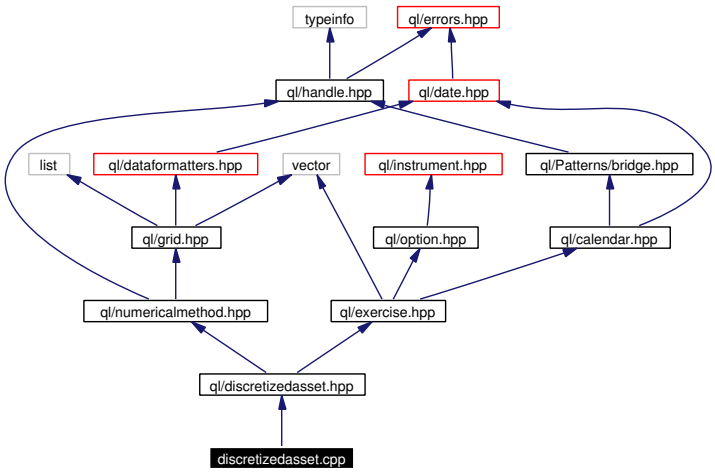
12.80 ql/discretizedasset.cpp File Reference

12.80.1 Detailed Description

Discretized asset classes.

`#include <ql/discretizedasset.hpp>`

Include dependency graph for discretizedasset.cpp:



Namespaces

- namespace **QuantLib**

12.81 ql/discretizedasset.hpp File Reference

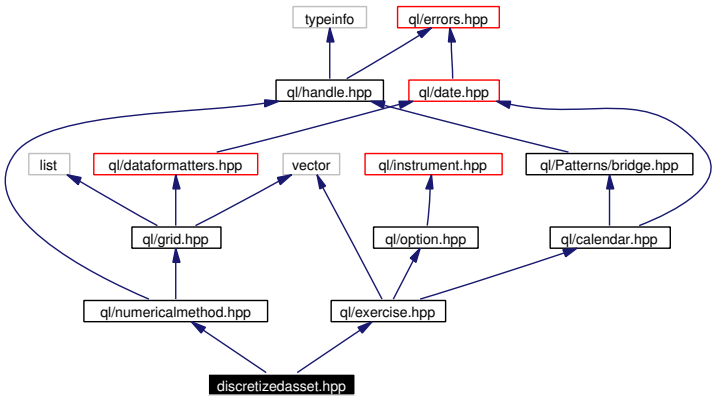
12.81.1 Detailed Description

Discretized asset classes.

`#include <ql/numericalmethod.hpp>`

`#include <ql/exercise.hpp>`

Include dependency graph for discretizedasset.hpp:



Namespaces

- namespace **QuantLib**

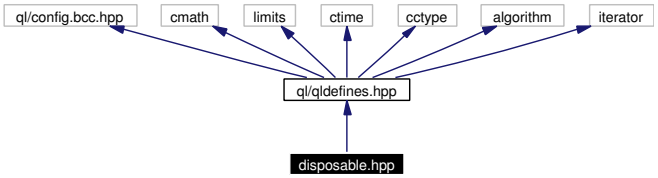
12.82 ql/disposable.hpp File Reference

12.82.1 Detailed Description

generic disposable object with move semantics

`#include <ql/qldefines.hpp>`

Include dependency graph for disposable.hpp:



Namespaces

- namespace **QuantLib**

12.83 ql/errors.hpp File Reference

12.83.1 Detailed Description

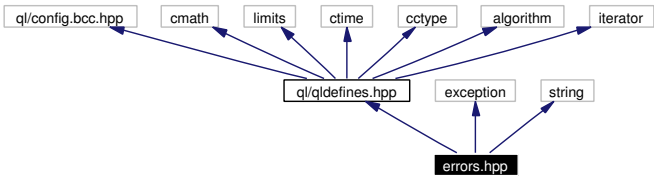
Classes and functions for error handling.

`#include <ql/qldefines.hpp>`

`#include <exception>`

`#include <string>`

Include dependency graph for errors.hpp:



Namespaces

- namespace **QuantLib**

Defines

- #define **QL_ASSERT**(condition, description)

  throw an error if the given condition is not verified

- #define **QL_REQUIRE**(condition, description)

  throw an error if the given pre-condition is not verified

- #define **QL_ENSURE**(condition, description)

  throw an error if the given post-condition is not verified

12.83.2 Define Documentation

12.83.2.1 #define QL_ASSERT(condition, description)

Value:

```
if (!(condition)) \
    throw QuantLib::AssertionFailedError(\
        QuantLib::Error::where(__FILE__,__LINE__) +  description); \
else
```

throw an error if the given condition is not verified

12.83.2.2 #define QL_REQUIRE(condition, description)

Value:

```
if (!(condition)) \
    throw QuantLib::PreconditionNotSatisfiedError(\
        QuantLib::Error::where(__FILE__,__LINE__) +  description); \
else
```

throw an error if the given pre-condition is not verified

Examples:
DiscreteHedging.cpp, and **swapvaluation.cpp**.

12.83.2.3 #define QL_ENSURE(condition, description)

Value:

```
if (!(condition)) \
    throw QuantLib::PostconditionNotSatisfiedError(\
        QuantLib::Error::where(__FILE__,__LINE__) +  description); \
else
```

throw an error if the given post-condition is not verified

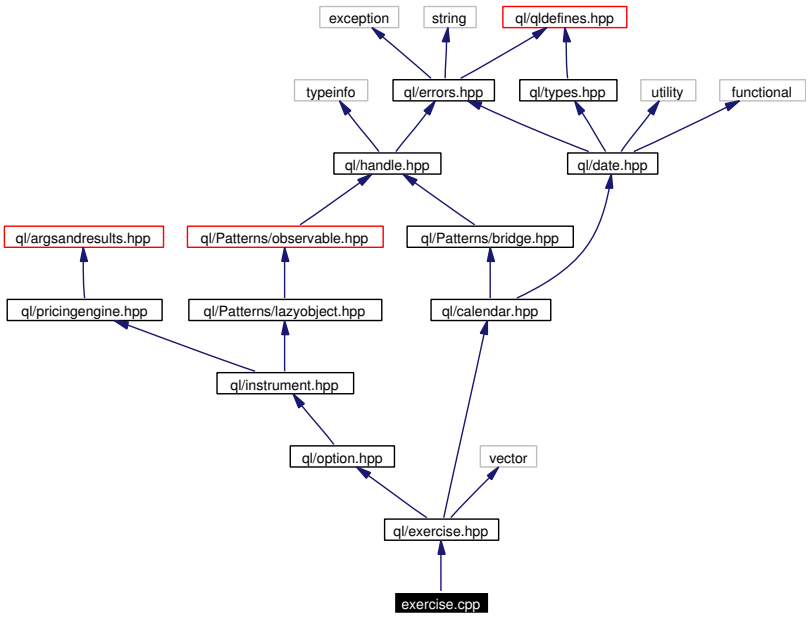
12.84 ql/exercise.cpp File Reference

12.84.1 Detailed Description

Option exercise classes and exercise payoff function.

`#include <ql/exercise.hpp>`

Include dependency graph for exercise.cpp:



Namespaces

- namespace **QuantLib**

12.85 ql/exercise.hpp File Reference

12.85.1 Detailed Description

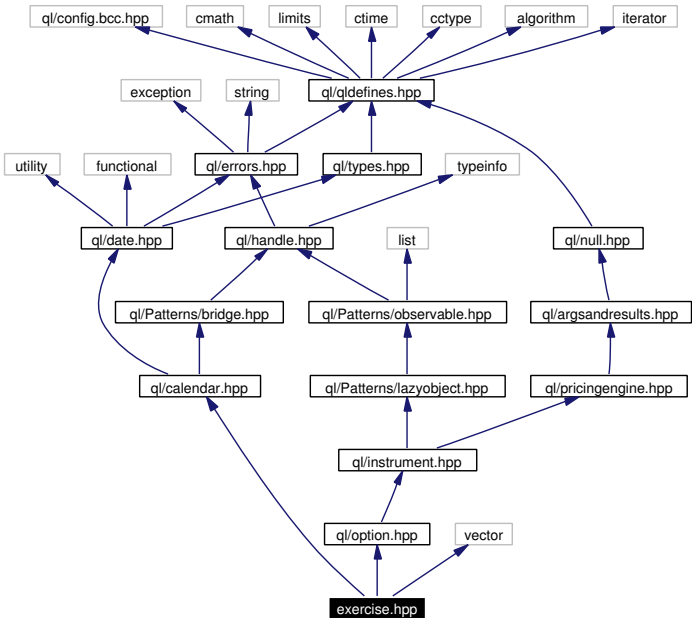
Option exercise classes and payoff function.

`#include <ql/calendar.hpp>`

`#include <ql/option.hpp>`

`#include <vector>`

Include dependency graph for exercise.hpp:



Namespaces

- namespace **QuantLib**

12.86 ql/FiniteDifferences/americancondition.hpp File Reference

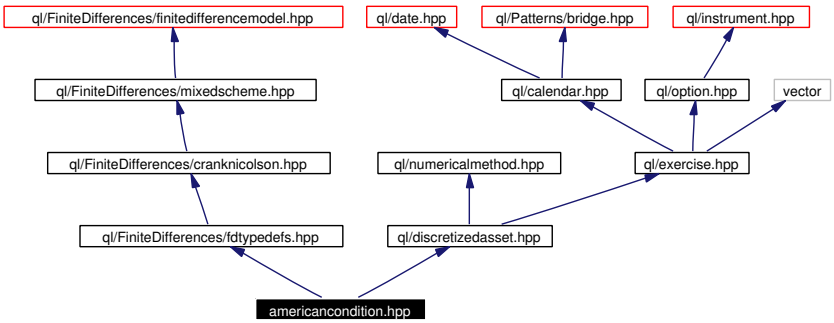
12.86.1 Detailed Description

american option exercise condition

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for americancondition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

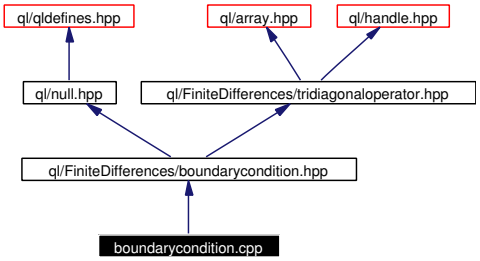
12.87 ql/FiniteDifferences/boundarycondition.cpp File Reference

12.87.1 Detailed Description

boundary conditions for differential operators

`#include <ql/FiniteDifferences/boundarycondition.hpp>`

Include dependency graph for boundarycondition.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.88 ql/FiniteDifferences/boundarycondition.hpp File Reference

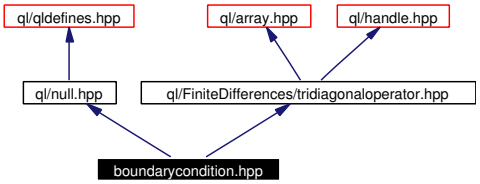
12.88.1 Detailed Description

boundary conditions for differential operators

`#include <ql/null.hpp>`

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for boundarycondition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

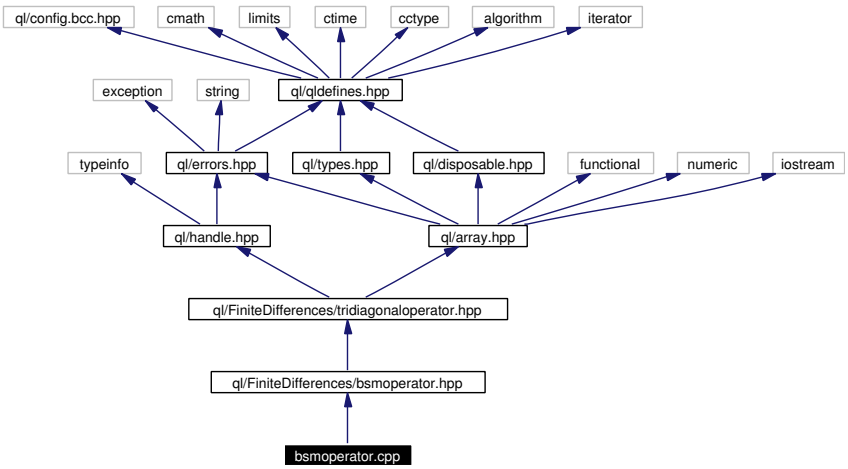
12.89 ql/FiniteDifferences/bsmoperator.cpp File Reference

12.89.1 Detailed Description

differential operator for Black-Scholes-Merton equation

`#include <ql/FiniteDifferences/bsmoperator.hpp>`

Include dependency graph for bsmoperator.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

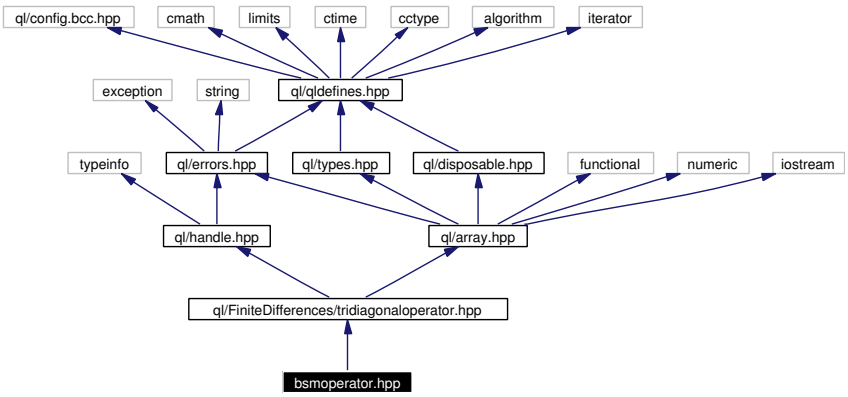
12.90 ql/FiniteDifferences/bsmoperator.hpp File Reference

12.90.1 Detailed Description

differential operator for Black-Scholes-Merton equation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for bsmoperator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

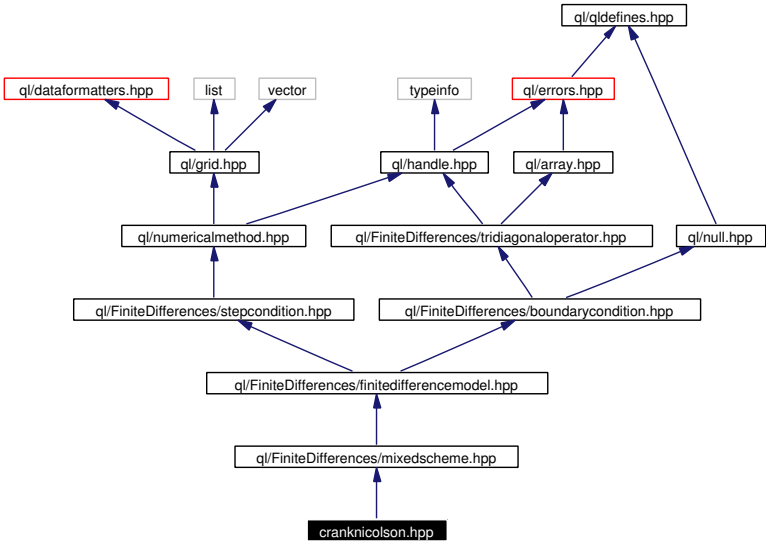
12.91 ql/FiniteDifferences/cranknicolson.hpp File Reference

12.91.1 Detailed Description

Crank-Nicolson scheme for finite difference methods.

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for cranknicolson.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

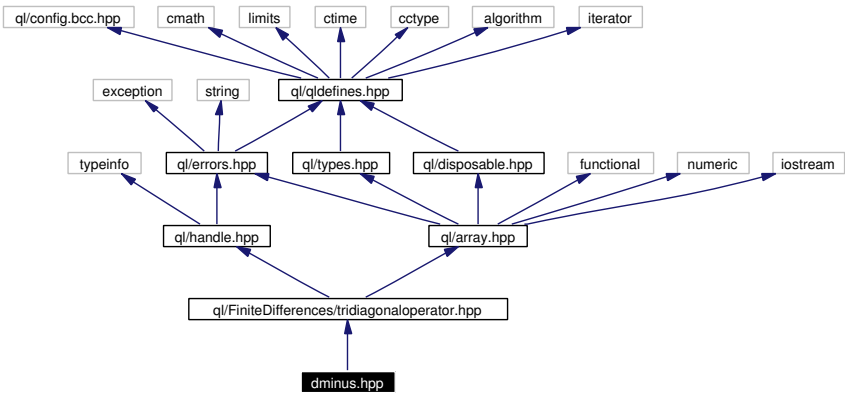
12.92 ql/FiniteDifferences/dminus.hpp File Reference

12.92.1 Detailed Description

D_- matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dminus.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

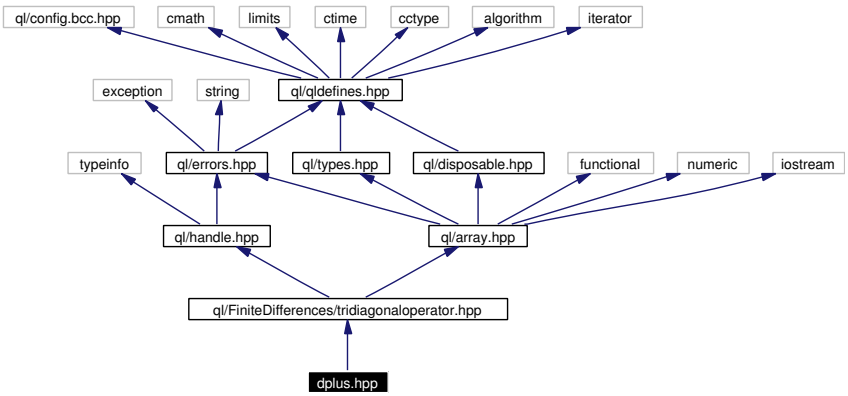
12.93 ql/FiniteDifferences/dplus.hpp File Reference

12.93.1 Detailed Description

D_+ matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dplus.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

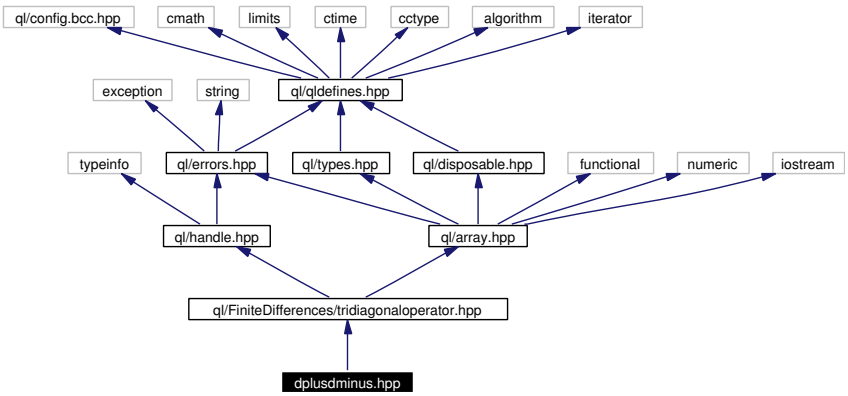
12.94 ql/FiniteDifferences/dplusdminus.hpp File Reference

12.94.1 Detailed Description

D_+D_- matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dplusdminus.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

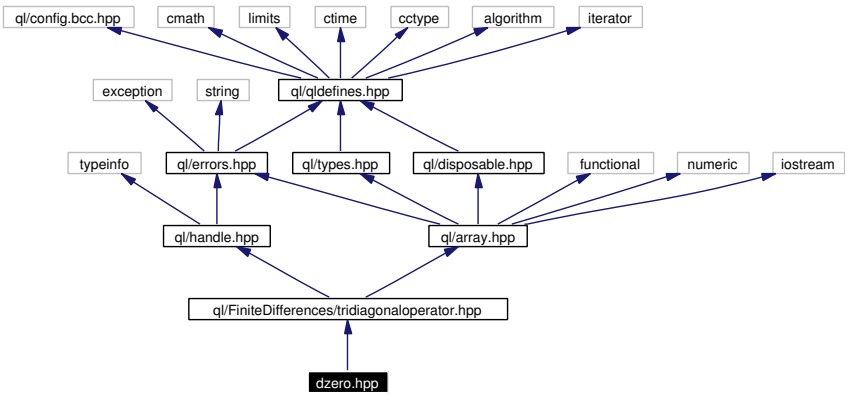
12.95 ql/FiniteDifferences/dzero.hpp File Reference

12.95.1 Detailed Description

D_0 matricial representation

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

Include dependency graph for dzero.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

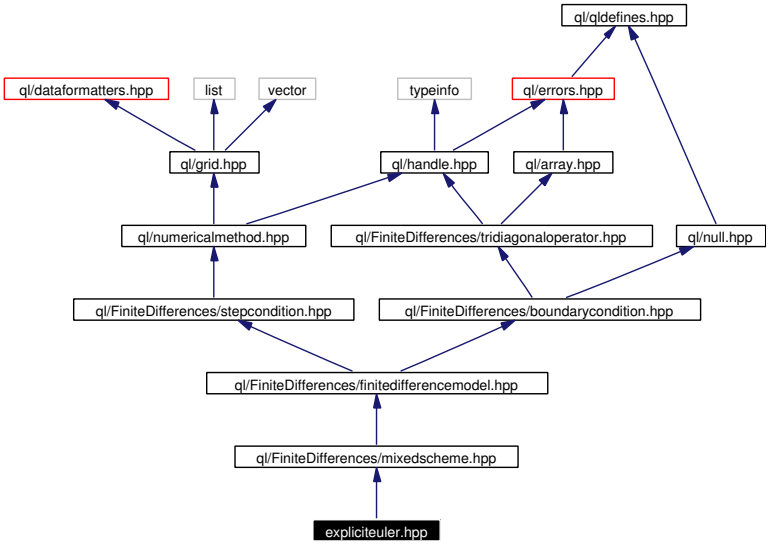
12.96 ql/FiniteDifferences/expliciteuler.hpp File Reference

12.96.1 Detailed Description

explicit Euler scheme for finite difference methods

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for expliciteuler.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

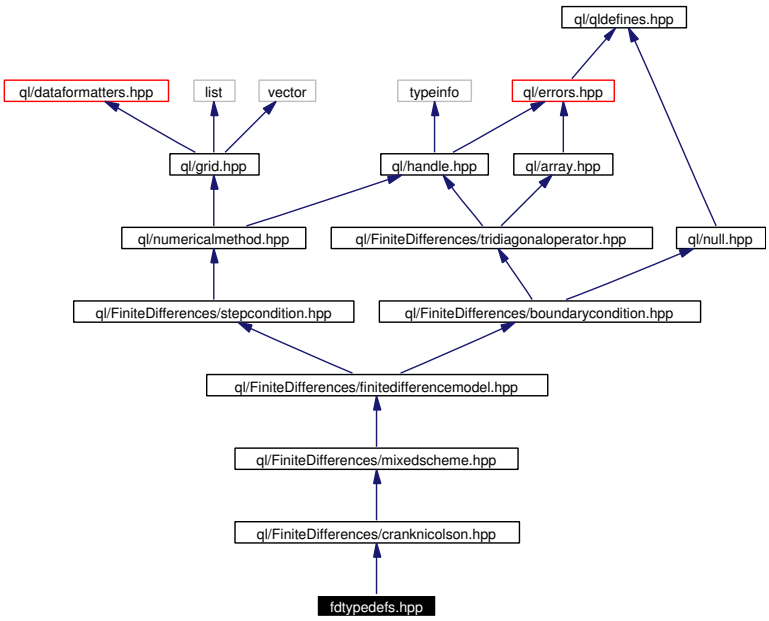
12.97 ql/FiniteDifferences/fdtypedefs.hpp File Reference

12.97.1 Detailed Description

default choices for template instantiations

`#include <ql/FiniteDifferences/cranknicolson.hpp>`

Include dependency graph for fdtypedefs.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.98 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

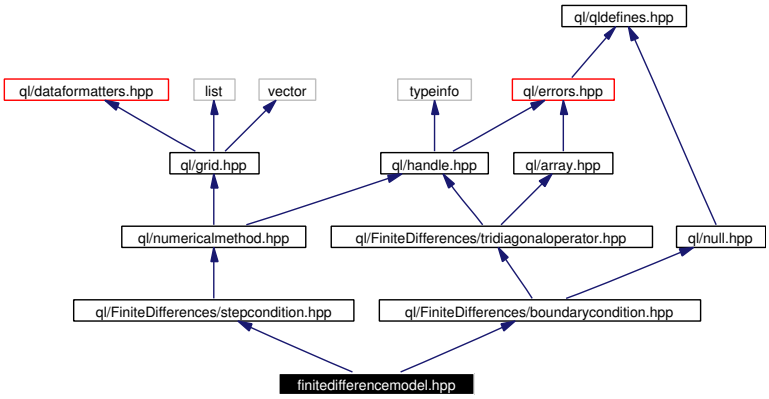
12.98.1 Detailed Description

generic finite difference model

`#include <ql/FiniteDifferences/stepcondition.hpp>`

`#include <ql/FiniteDifferences/boundarycondition.hpp>`

Include dependency graph for finitedifferencemodel.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

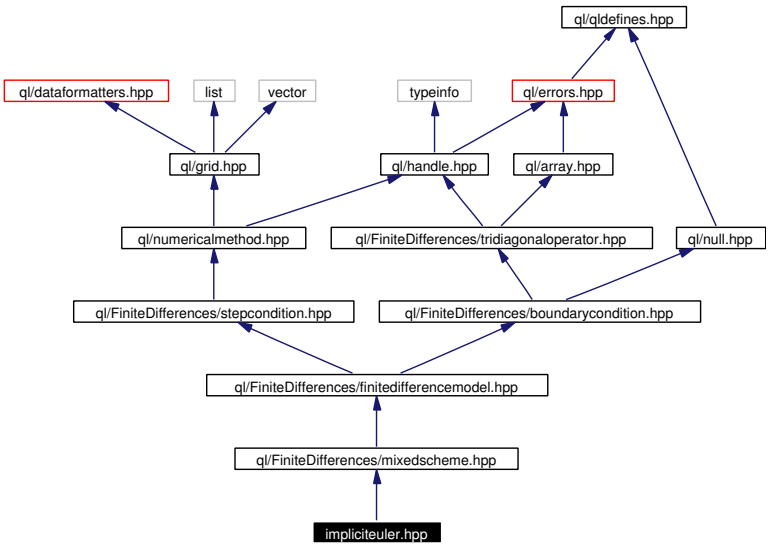
12.99 ql/FiniteDifferences/impliciteuler.hpp File Reference

12.99.1 Detailed Description

implicit Euler scheme for finite difference methods

`#include <ql/FiniteDifferences/mixedscheme.hpp>`

Include dependency graph for impliciteuler.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

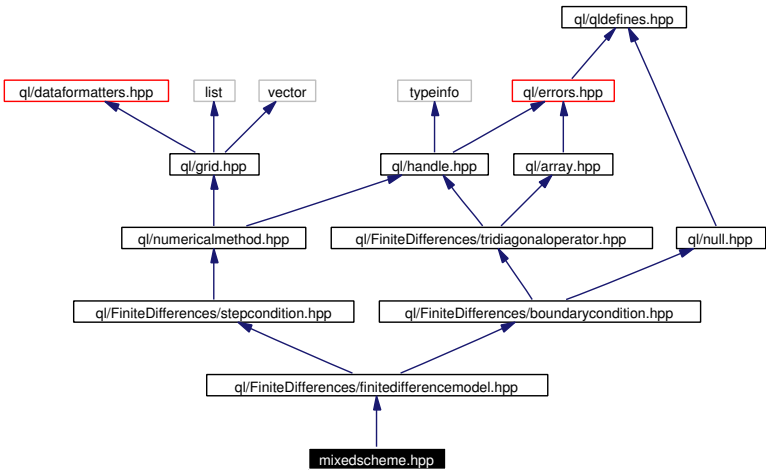
12.100 ql/FiniteDifferences/mixedscheme.hpp File Reference

12.100.1 Detailed Description

Mixed (explicit/implicit) scheme for finite difference methods.

`#include <ql/FiniteDifferences/finitedifferencemodel.hpp>`

Include dependency graph for mixedscheme.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

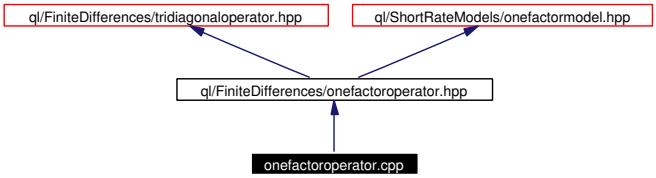
12.101 ql/FiniteDifferences/onefactoroperator.cpp File Reference

12.101.1 Detailed Description

differential operator for one-factor interest rate models

`#include "ql/FiniteDifferences/onefactoroperator.hpp"`

Include dependency graph for onefactoroperator.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.102 ql/FiniteDifferences/onefactoroperator.hpp File Reference

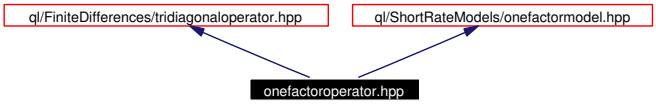
12.102.1 Detailed Description

general differential operator for one-factor interest rate models

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for onefactoroperator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.103 ql/FiniteDifferences/shoutcondition.hpp File Reference

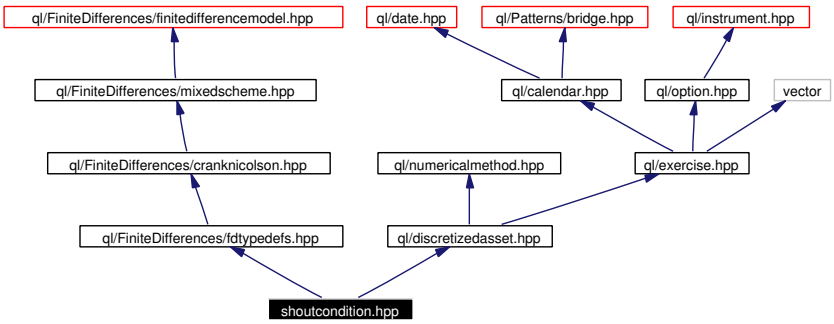
12.103.1 Detailed Description

shout option exercise condition

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for shoutcondition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

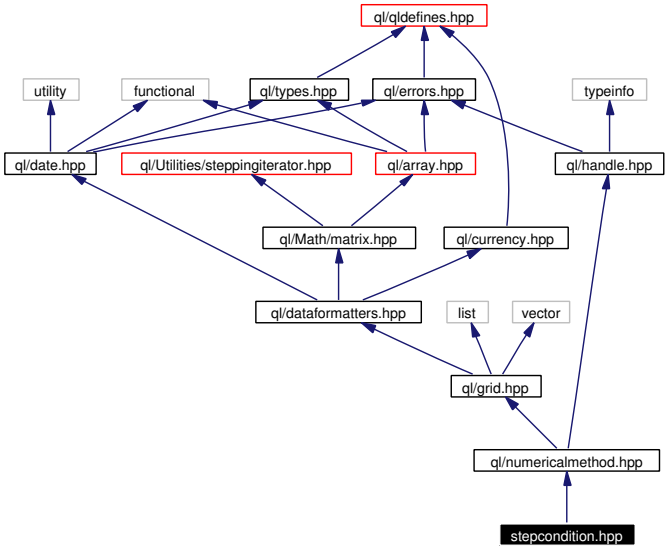
12.104 ql/FiniteDifferences/stepcondition.hpp File Reference

12.104.1 Detailed Description

conditions to be applied at every time step

`#include <ql/numericalmethod.hpp>`

Include dependency graph for stepcondition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.105 ql/FiniteDifferences/tridiagonaloperator.cpp File Reference

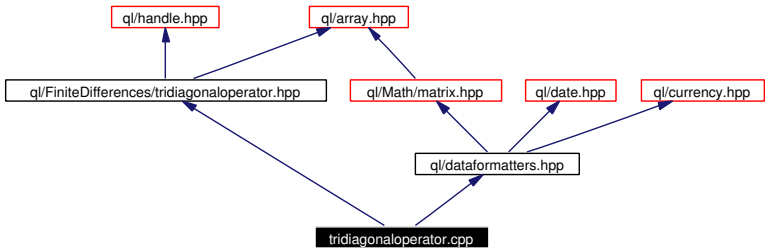
12.105.1 Detailed Description

tridiagonal operator

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for tridiagonaloperator.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

12.106 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

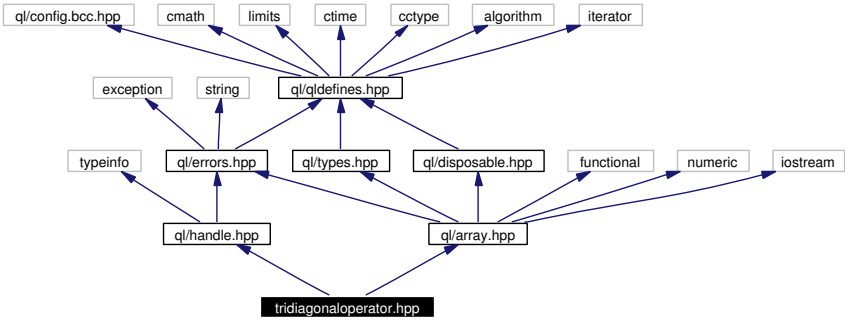
12.106.1 Detailed Description

tridiagonal operator

`#include <ql/array.hpp>`

`#include <ql/handle.hpp>`

Include dependency graph for tridiagonaloperator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

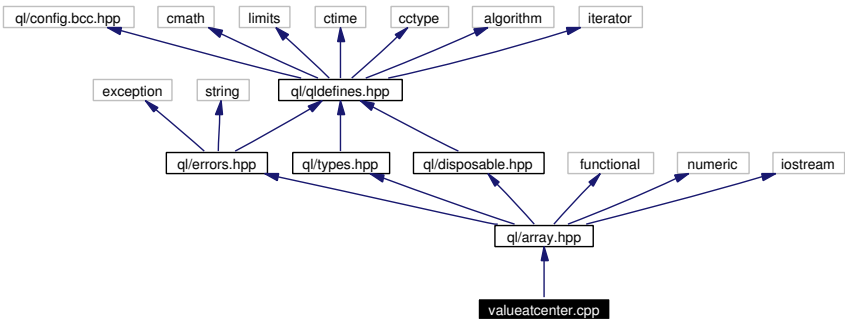
12.107 ql/FiniteDifferences/valueatcenter.cpp File Reference

12.107.1 Detailed Description

compute value, first, and second derivatives at grid center

`#include <ql/array.hpp>`

Include dependency graph for valueatcenter.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

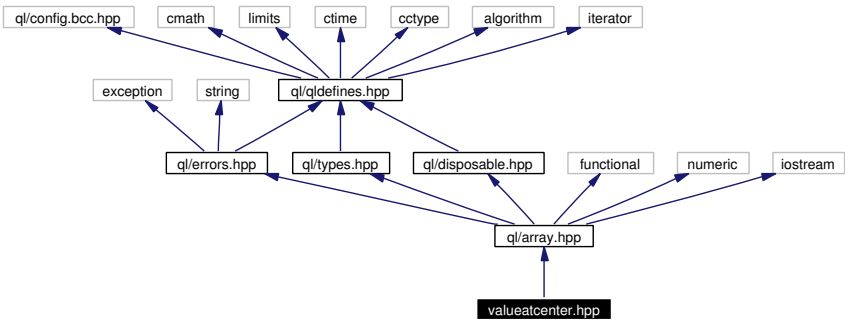
12.108 ql/FiniteDifferences/valueatcenter.hpp File Reference

12.108.1 Detailed Description

compute value, first, and second derivatives at grid center

`#include <ql/array.hpp>`

Include dependency graph for valueatcenter.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::FiniteDifferences**

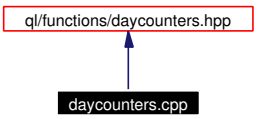
12.109 ql/functions/daycounters.cpp File Reference

12.109.1 Detailed Description

day counters functions

`#include <ql/functions/daycounters.hpp>`

Include dependency graph for daycounters.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

12.110 ql/functions/daycounters.hpp File Reference

12.110.1 Detailed Description

day counters functions

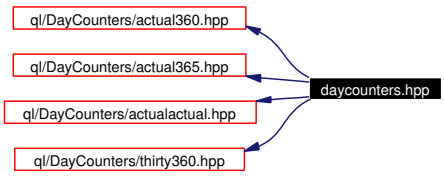
`#include <ql/DayCounters/actual360.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

`#include <ql/DayCounters/actualactual.hpp>`

`#include <ql/DayCounters/thirty360.hpp>`

Include dependency graph for daycounters.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

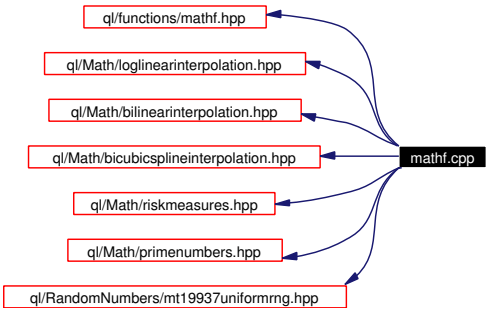
12.111 ql/functions/mathf.cpp File Reference

12.111.1 Detailed Description

math functions

```
#include <ql/functions/mathf.hpp>
#include <ql/Math/loglinearinterpolation.hpp>
#include <ql/Math/bilinearinterpolation.hpp>
#include <ql/Math/bicubicsplineinterpolation.hpp>
#include <ql/Math/riskmeasures.hpp>
#include <ql/Math/primenumbers.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

Include dependency graph for mathf.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

12.112 ql/functions/mathf.hpp File Reference

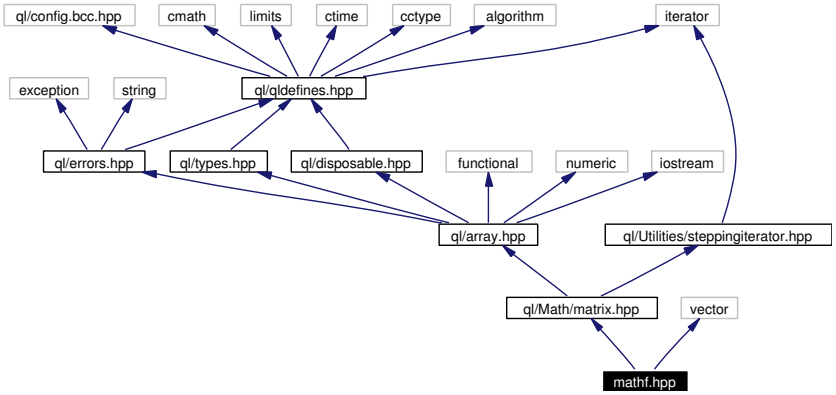
12.112.1 Detailed Description

math functions

`#include <ql/Math/matrix.hpp>`

`#include <vector>`

Include dependency graph for mathf.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

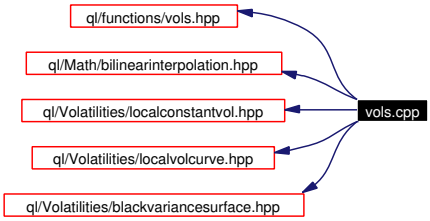
12.113 ql/functions/vols.cpp File Reference

12.113.1 Detailed Description

Volatility functions.

```
#include <ql/functions/vols.hpp>
#include <ql/Math/bilinearinterpolation.hpp>
#include <ql/Volatilities/localconstantvol.hpp>
#include <ql/Volatilities/localvolcurve.hpp>
#include <ql/Volatilities/blackvariancesurface.hpp>
```

Include dependency graph for vols.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

12.114 ql/functions/vols.hpp File Reference

12.114.1 Detailed Description

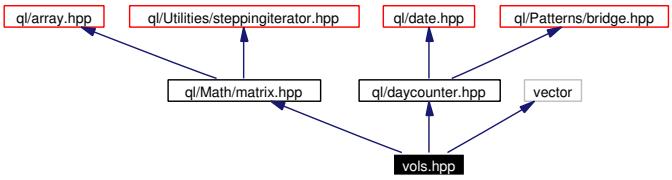
Volatility functions.

`#include <ql/Math/matrix.hpp>`

`#include <ql/daycounter.hpp>`

`#include <vector>`

Include dependency graph for vols.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Functions**

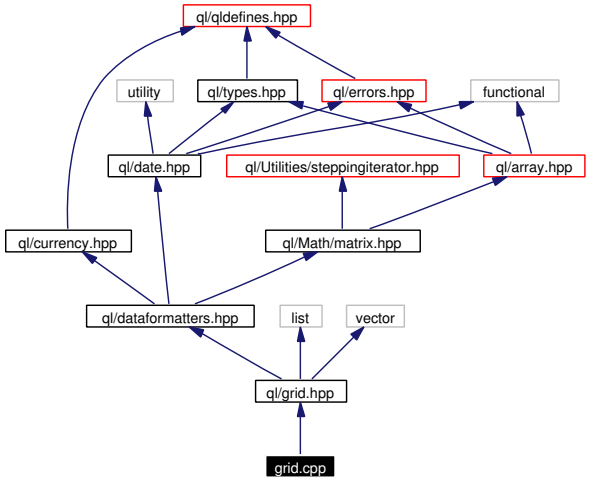
12.115 ql/grid.cpp File Reference

12.115.1 Detailed Description

Grid classes with useful constructors for trees and finite diffs.

`#include <ql/grid.hpp>`

Include dependency graph for grid.cpp:



Namespaces

- namespace **QuantLib**

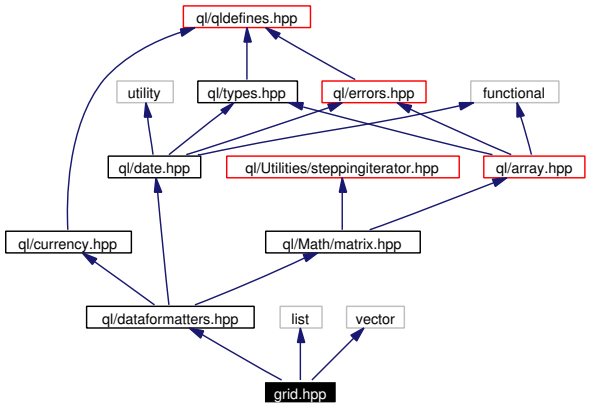
12.116 ql/grid.hpp File Reference

12.116.1 Detailed Description

Grid classes with useful constructors for trees and finite diffs.

```
#include <ql/dataformatters.hpp>
#include <list>
#include <vector>
```

Include dependency graph for grid.hpp:



Namespaces

- namespace **QuantLib**

12.117 ql/handle.hpp File Reference

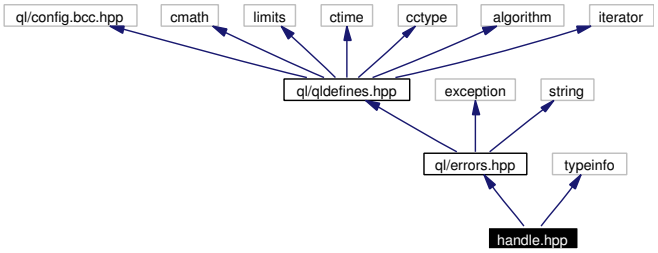
12.117.1 Detailed Description

Reference-counted pointer.

`#include <ql/errors.hpp>`

`#include <typeinfo>`

Include dependency graph for handle.hpp:



Namespaces

- namespace **QuantLib**

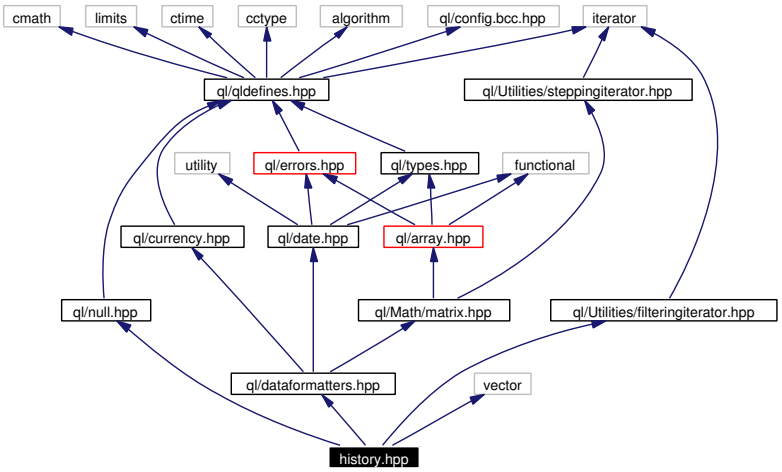
12.118 ql/history.hpp File Reference

12.118.1 Detailed Description

history class

```
#include <ql/null.hpp>
#include <ql/Utilities/filteringiterator.hpp>
#include <ql/dataformatters.hpp>
#include <vector>
```

Include dependency graph for history.hpp:



Namespaces

- namespace **QuantLib**

12.119 ql/index.hpp File Reference

12.119.1 Detailed Description

purely virtual base class for indexes

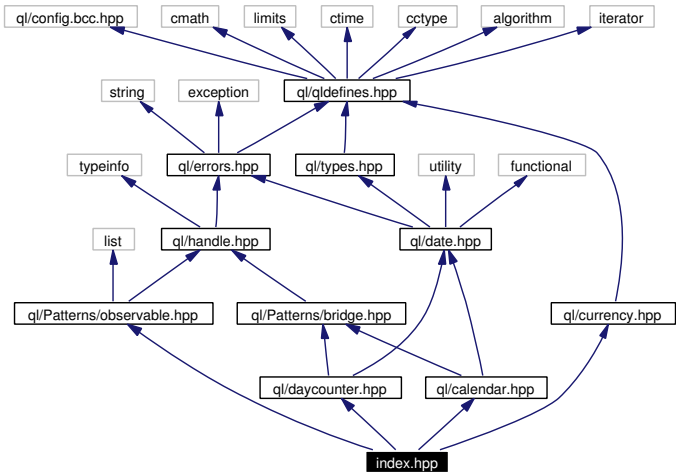
`#include <ql/calendar.hpp>`

`#include <ql/currency.hpp>`

`#include <ql/daycounter.hpp>`

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for index.hpp:



Namespaces

- namespace **QuantLib::Indexes**
- namespace **QuantLib**

12.120 ql/Indexes/audlibor.hpp File Reference

12.120.1 Detailed Description

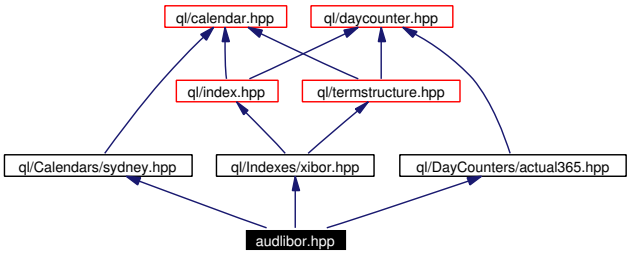
AUD Libor index (check settlement days)

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/sydney.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

Include dependency graph for audlibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.121 ql/Indexes/cadlibor.hpp File Reference

12.121.1 Detailed Description

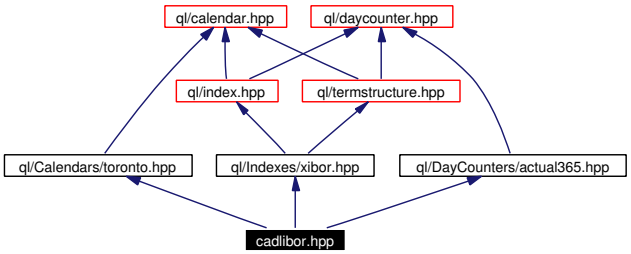
CAD Libor index (Also known as CDOR)

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/toronto.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

Include dependency graph for cadlibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.122 ql/Indexes/chflibor.hpp File Reference

12.122.1 Detailed Description

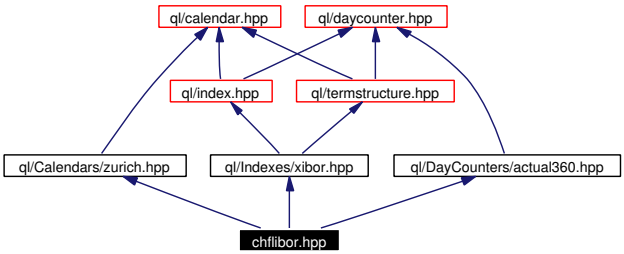
CHF Libor index (Also known as ZIBOR)

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/zurich.hpp>`

`#include <ql/DayCounters/actual360.hpp>`

Include dependency graph for chflibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.123 ql/Indexes/euribor.hpp File Reference

12.123.1 Detailed Description

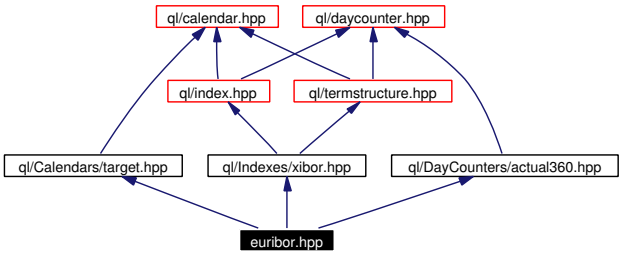
Euribor index

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/target.hpp>`

`#include <ql/DayCounters/actual360.hpp>`

Include dependency graph for euribor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.124 ql/Indexes/gbplibor.hpp File Reference

12.124.1 Detailed Description

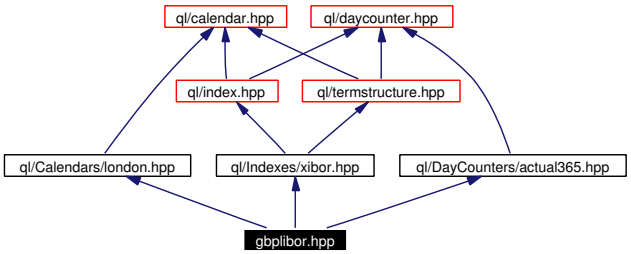
GBP Libor index

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/london.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

Include dependency graph for gbplibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.125 ql/Indexes/jpylibor.hpp File Reference

12.125.1 Detailed Description

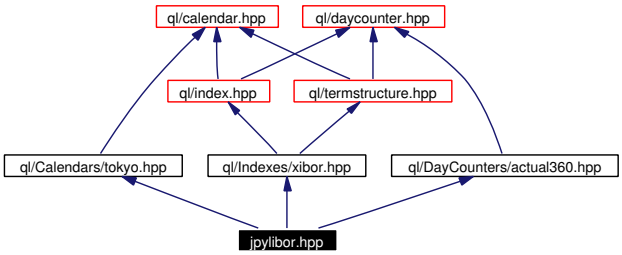
JPY Libor index (Also known as TIBOR, check settlement days)

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/tokyo.hpp>`

`#include <ql/DayCounters/actual360.hpp>`

Include dependency graph for jpylibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.126 ql/Indexes/usdlibor.hpp File Reference

12.126.1 Detailed Description

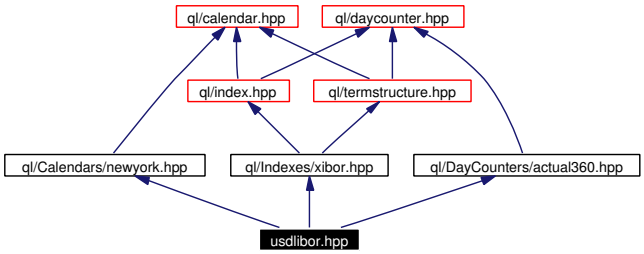
USD Libor index

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/newyork.hpp>`

`#include <ql/DayCounters/actual360.hpp>`

Include dependency graph for usdlibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.127 ql/Indexes/xibor.cpp File Reference

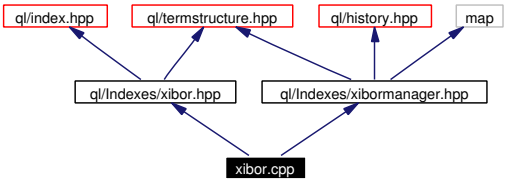
12.127.1 Detailed Description

purely virtual base class for libor indexes

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Indexes/xibormanager.hpp>`

Include dependency graph for xibor.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.128 ql/Indexes/xibor.hpp File Reference

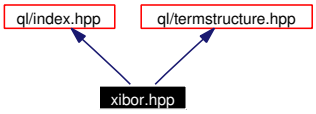
12.128.1 Detailed Description

base class for libor indexes

`#include <ql/index.hpp>`

`#include <ql/termstructure.hpp>`

Include dependency graph for xibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

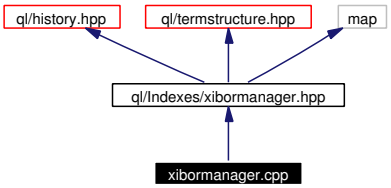
12.129 ql/Indexes/xibormanager.cpp File Reference

12.129.1 Detailed Description

global repository for Xibor histories

`#include <ql/Indexes/xibormanager.hpp>`

Include dependency graph for xibormanager.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

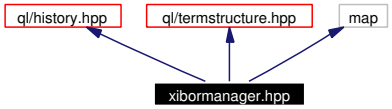
12.130 ql/Indexes/xibormanager.hpp File Reference

12.130.1 Detailed Description

global repository for Xibor histories

```
#include <ql/history.hpp>
#include <ql/termstructure.hpp>
#include <map>
```

Include dependency graph for xibormanager.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.131 ql/Indexes/zarlibor.hpp File Reference

12.131.1 Detailed Description

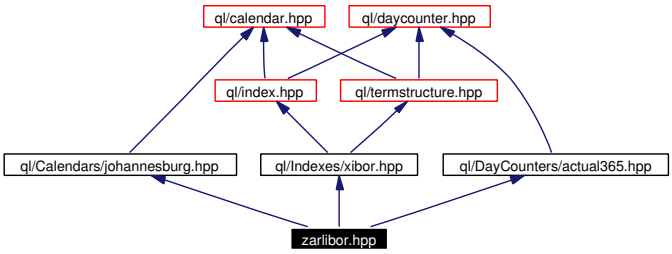
ZAR Libor index (also known as JIBAR)

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/Calendars/johannesburg.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

Include dependency graph for zarlibor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Indexes**

12.132 ql/instrument.hpp File Reference

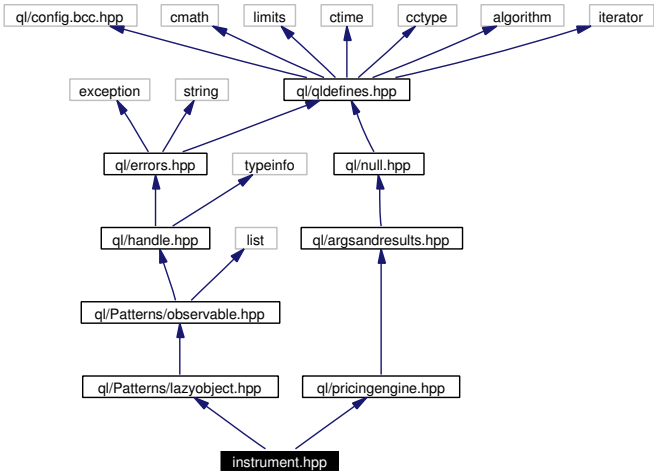
12.132.1 Detailed Description

Abstract instrument class.

`#include <ql/Patterns/lazyobject.hpp>`

`#include <ql/pricingengine.hpp>`

Include dependency graph for instrument.hpp:



Namespaces

- namespace **QuantLib::Instruments**
- namespace **QuantLib**

12.133 ql/Instruments/barrieroption.cpp File Reference

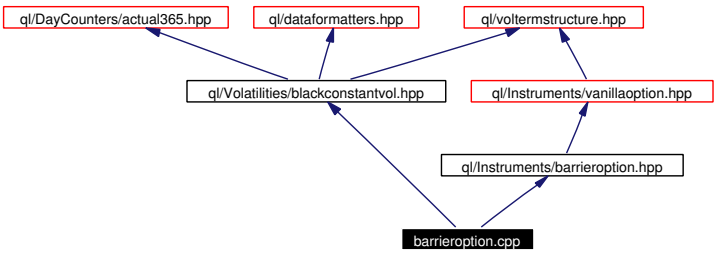
12.133.1 Detailed Description

Barrier option on a single asset.

`#include <ql/Volatilities/blackconstantvol.hpp>`

`#include <ql/Instruments/barrieroption.hpp>`

Include dependency graph for barrieroption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

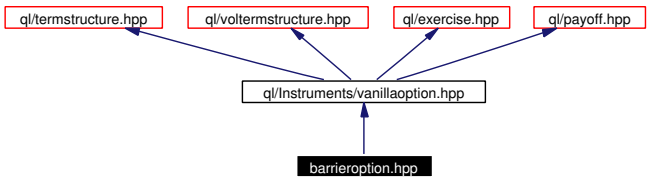
12.134 ql/Instruments/barrieroption.hpp File Reference

12.134.1 Detailed Description

Barrier option on a single asset.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for barrieroption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.135 ql/Instruments/binaryoption.cpp File Reference

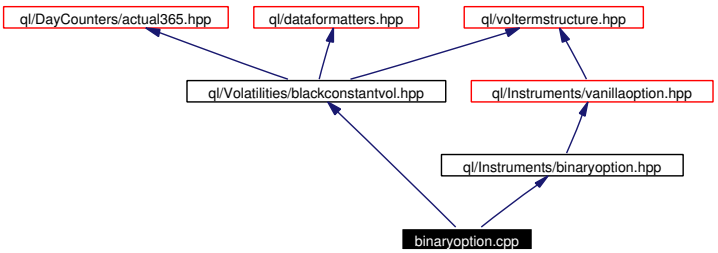
12.135.1 Detailed Description

Binary option on a single asset.

`#include <ql/Volatilities/blackconstantvol.hpp>`

`#include <ql/Instruments/binaryoption.hpp>`

Include dependency graph for binaryoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

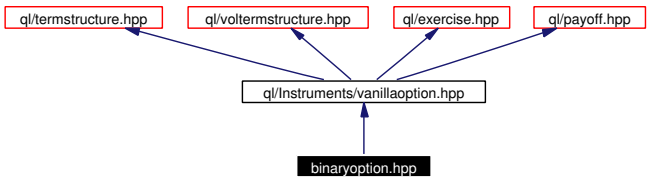
12.136 ql/Instruments/binaryoption.hpp File Reference

12.136.1 Detailed Description

Binary option on a single asset.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for binaryoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

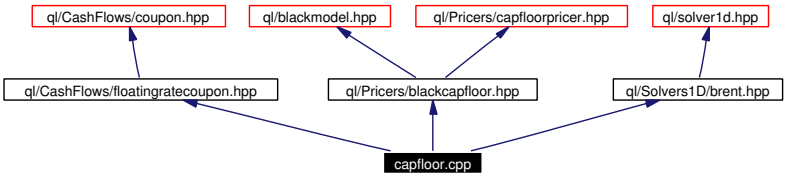
12.137 ql/Instruments/capfloor.cpp File Reference

12.137.1 Detailed Description

European cap and floor class.

```
#include <ql/CashFlows/floatingratecoupon.hpp>
#include <ql/Pricers/blackcapfloor.hpp>
#include <ql/Solvers1D/brent.hpp>
```

Include dependency graph for capfloor.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.138 ql/Instruments/capfloor.hpp File Reference

12.138.1 Detailed Description

Cap and Floor class.

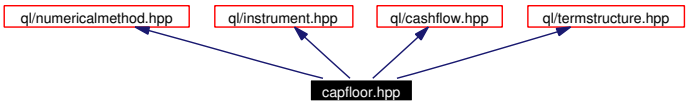
`#include <ql/numericalmethod.hpp>`

`#include <ql/instrument.hpp>`

`#include <ql/cashflow.hpp>`

`#include <ql/termstructure.hpp>`

Include dependency graph for capfloor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

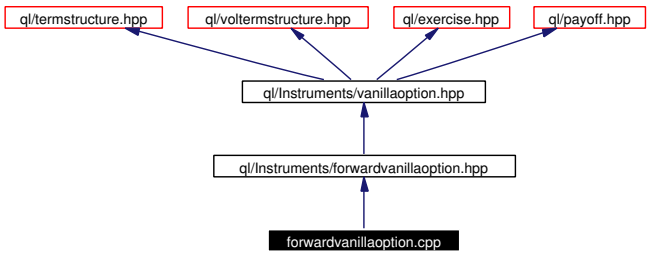
12.139 ql/Instruments/forwardvanillaoption.cpp File Reference

12.139.1 Detailed Description

Forward version of a vanilla option.

`#include <ql/Instruments/forwardvanillaoption.hpp>`

Include dependency graph for forwardvanillaoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

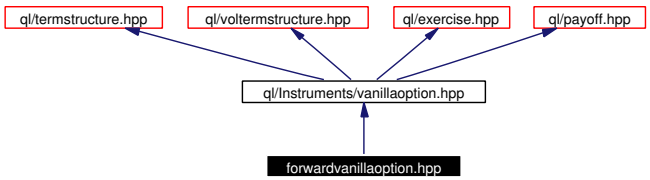
12.140 ql/Instruments/forwardvanillaoption.hpp File Reference

12.140.1 Detailed Description

Forward version of a vanilla option.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for forwardvanillaoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

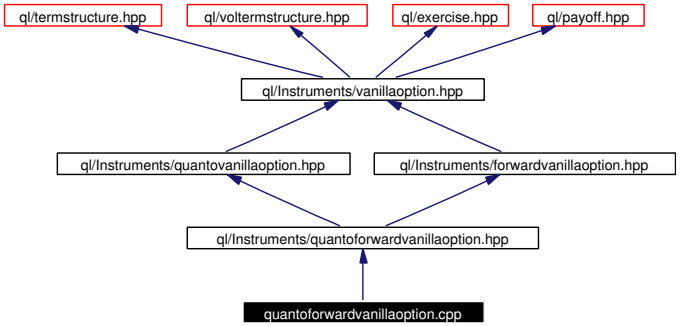
12.141 ql/Instruments/quantoforwardvanillaoption.cpp File Reference

12.141.1 Detailed Description

Quanto version of a forward vanilla option.

`#include <ql/Instruments/quantoforwardvanillaoption.hpp>`

Include dependency graph for quantoforwardvanillaoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.142 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

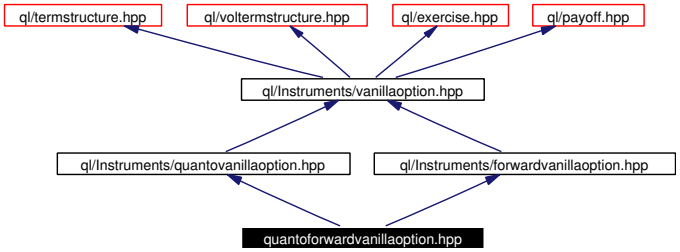
12.142.1 Detailed Description

Quanto version of a forward vanilla option.

`#include <ql/Instruments/quantovanillaoption.hpp>`

`#include <ql/Instruments/forwardvanillaoption.hpp>`

Include dependency graph for quantoforwardvanillaoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

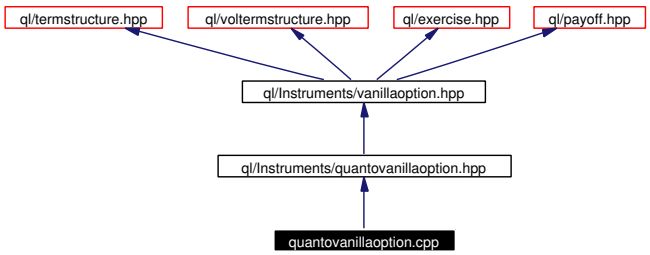
12.143 ql/Instruments/quantovanillaoption.cpp File Reference

12.143.1 Detailed Description

Quanto version of a vanilla option.

`#include <ql/Instruments/quantovanillaoption.hpp>`

Include dependency graph for quantovanillaoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

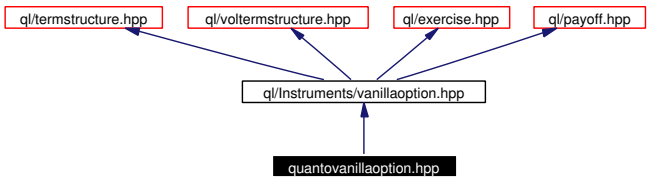
12.144 ql/Instruments/quantovanillaoption.hpp File Reference

12.144.1 Detailed Description

Quanto version of a vanilla option.

`#include <ql/Instruments/vanillaoption.hpp>`

Include dependency graph for quantovanillaoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

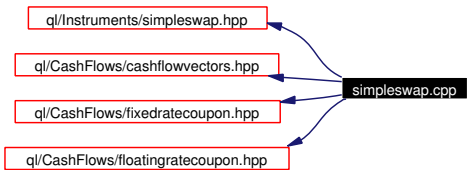
12.145 ql/Instruments/simpleswap.cpp File Reference

12.145.1 Detailed Description

Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/simpleswap.hpp>
#include <ql/CashFlows/cashflowvectors.hpp>
#include <ql/CashFlows/fixedratecoupon.hpp>
#include <ql/CashFlows/floatingratecoupon.hpp>
```

Include dependency graph for simpleswap.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.146 ql/Instruments/simpleswap.hpp File Reference

12.146.1 Detailed Description

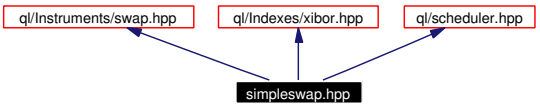
Simple fixed-rate vs Libor swap.

`#include <ql/Instruments/swap.hpp>`

`#include <ql/Indexes/xibor.hpp>`

`#include <ql/scheduler.hpp>`

Include dependency graph for simpleswap.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

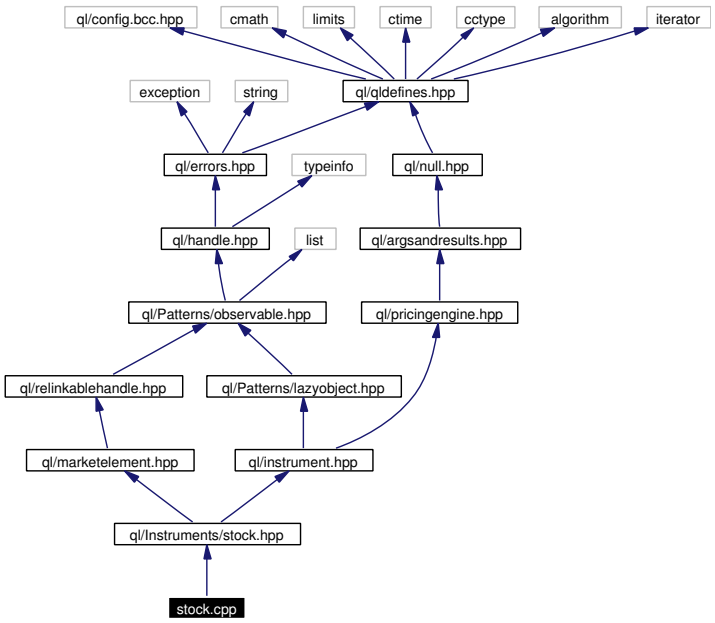
12.147 ql/Instruments/stock.cpp File Reference

12.147.1 Detailed Description

concrete stock class

`#include <ql/Instruments/stock.hpp>`

Include dependency graph for stock.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.148 ql/Instruments/stock.hpp File Reference

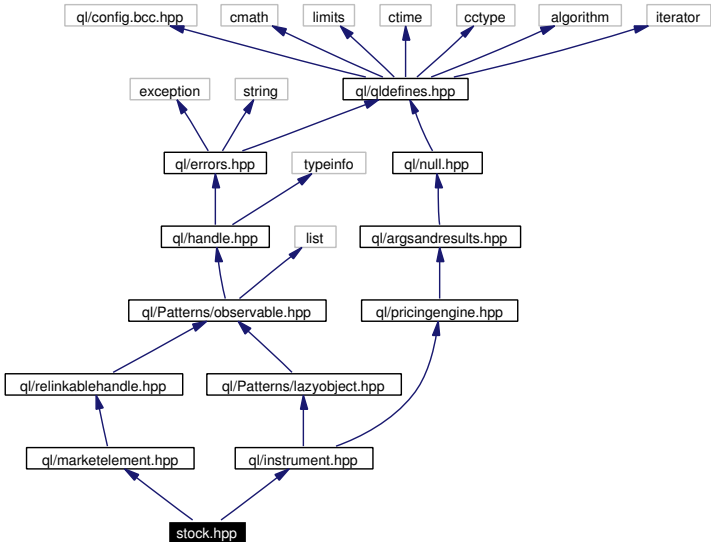
12.148.1 Detailed Description

concrete stock class

`#include <ql/instrument.hpp>`

`#include <ql/marketelement.hpp>`

Include dependency graph for stock.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.149 ql/Instruments/swap.cpp File Reference

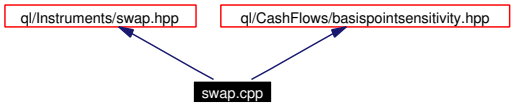
12.149.1 Detailed Description

Interest rate swap.

`#include <ql/Instruments/swap.hpp>`

`#include <ql/CashFlows/basispointsensitivity.hpp>`

Include dependency graph for swap.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

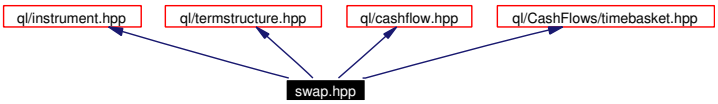
12.150 ql/Instruments/swap.hpp File Reference

12.150.1 Detailed Description

Interest rate swap.

```
#include <ql/instrument.hpp>
#include <ql/termstructure.hpp>
#include <ql/cashflow.hpp>
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for swap.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

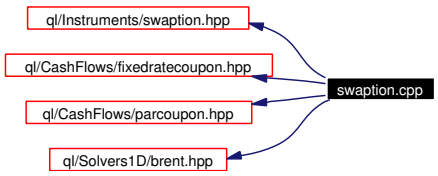
12.151 ql/Instruments/swaption.cpp File Reference

12.151.1 Detailed Description

Swaption.

```
#include <ql/Instruments/swaption.hpp>
#include <ql/CashFlows/fixedratecoupon.hpp>
#include <ql/CashFlows/parcoupon.hpp>
#include <ql/Solvers1D/brent.hpp>
```

Include dependency graph for swaption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.152 ql/Instruments/swaption.hpp File Reference

12.152.1 Detailed Description

Swaption class.

`#include <ql/exercise.hpp>`

`#include <ql/numericalmethod.hpp>`

`#include <ql/Instruments/simpleswap.hpp>`

Include dependency graph for swaption.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

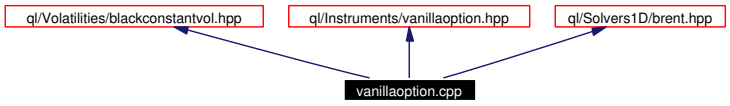
12.153 ql/Instruments/vanillaoption.cpp File Reference

12.153.1 Detailed Description

Vanilla option on a single asset.

```
#include <ql/Volatilities/blackconstantvol.hpp>
#include <ql/Instruments/vanillaoption.hpp>
#include <ql/Solvers1D/brent.hpp>
```

Include dependency graph for vanillaoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

12.154 ql/Instruments/vanillaoption.hpp File Reference

12.154.1 Detailed Description

Vanilla option on a single asset.

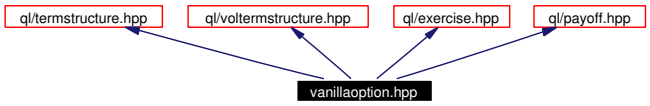
`#include <ql/termstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

`#include <ql/exercise.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for vanillaoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Instruments**

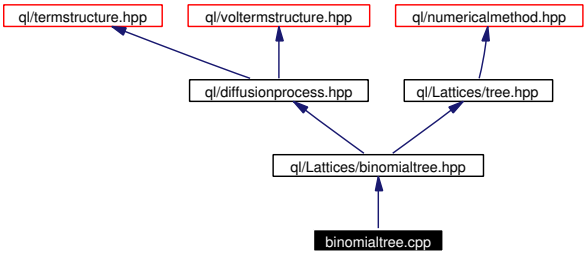
12.155 ql/Lattices/binomialtree.cpp File Reference

12.155.1 Detailed Description

Binomial tree class.

`#include "ql/Lattices/binomialtree.hpp"`

Include dependency graph for binomialtree.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

12.156 ql/Lattices/binomialtree.hpp File Reference

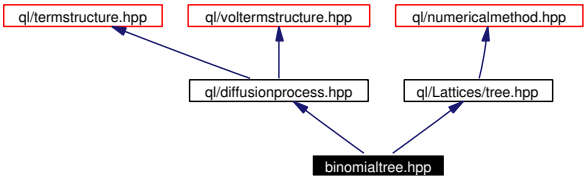
12.156.1 Detailed Description

Binomial tree class.

`#include <ql/diffusionprocess.hpp>`

`#include <ql/Lattices/tree.hpp>`

Include dependency graph for binomialtree.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

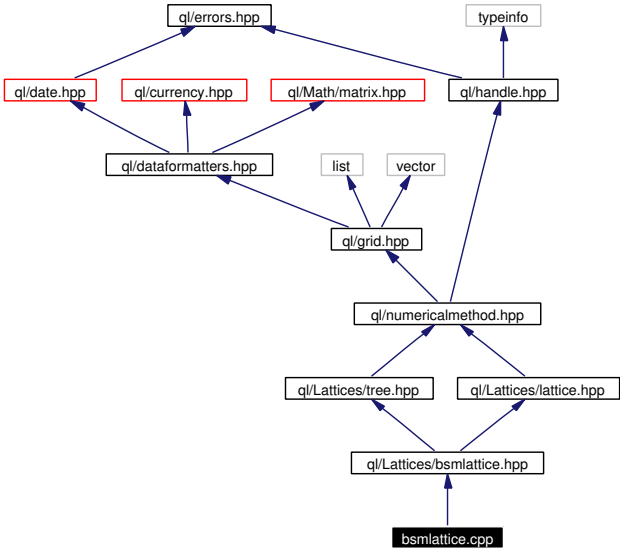
12.157 ql/Lattices/bsmlattice.cpp File Reference

12.157.1 Detailed Description

Binomial trees under the BSM model.

`#include "ql/Lattices/bsmlattice.hpp"`

Include dependency graph for bsmlattice.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

12.158 ql/Lattices/bsmlattice.hpp File Reference

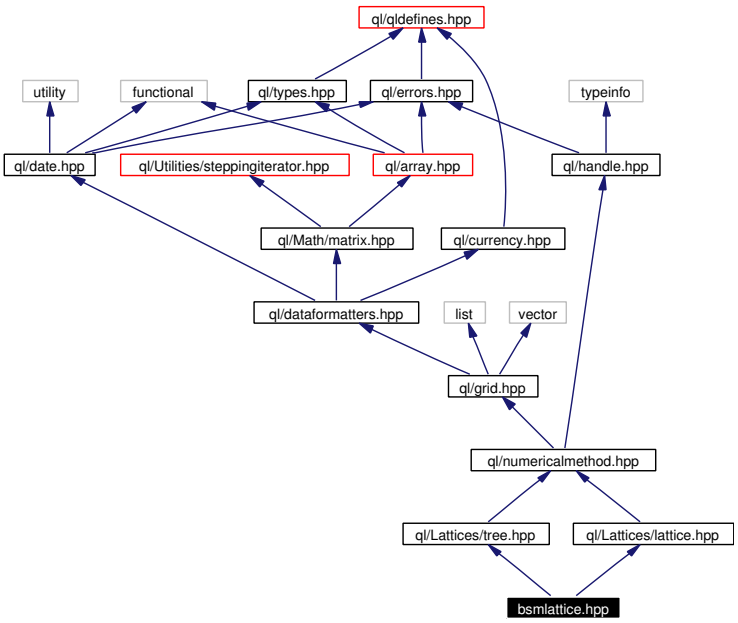
12.158.1 Detailed Description

Binomial trees under the BSM model.

`#include <ql/Lattices/tree.hpp>`

`#include <ql/Lattices/lattice.hpp>`

Include dependency graph for bsmlattice.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

12.159 ql/Lattices/lattice.cpp File Reference

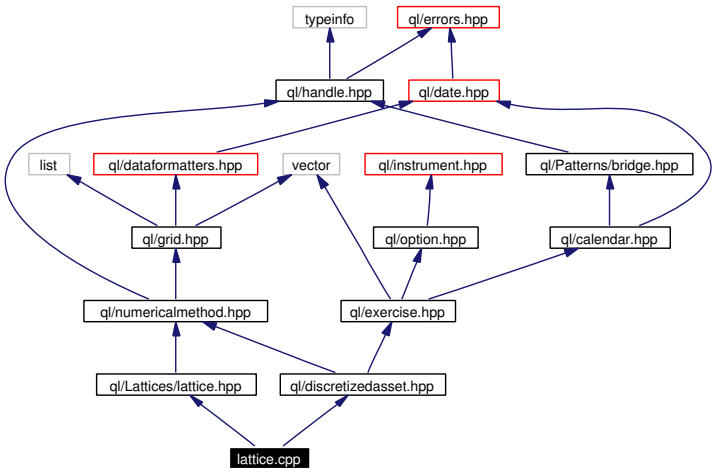
12.159.1 Detailed Description

Lattice class.

`#include <ql/Lattices/lattice.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for lattice.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

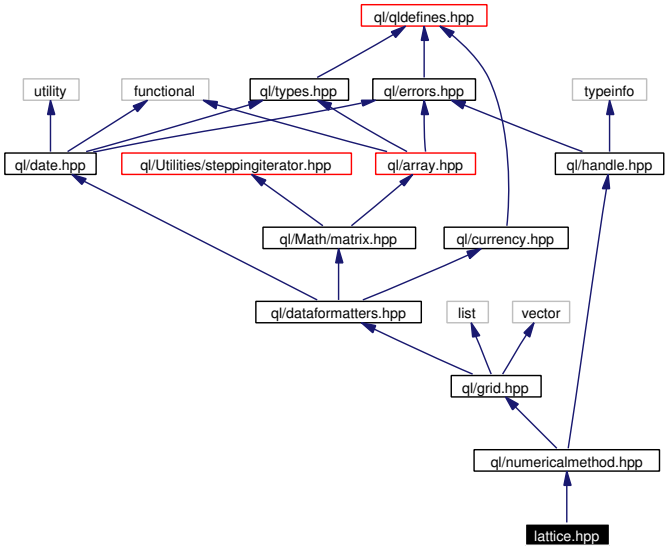
12.160 ql/Lattices/lattice.hpp File Reference

12.160.1 Detailed Description

Lattice method class.

`#include <ql/numericalmethod.hpp>`

Include dependency graph for lattice.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

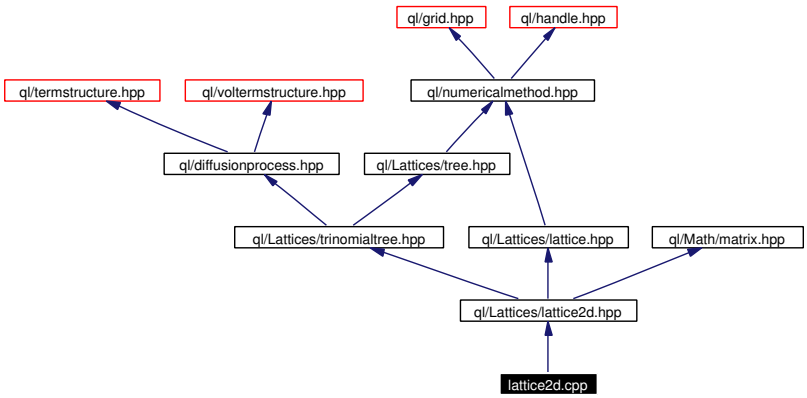
12.161 ql/Lattices/lattice2d.cpp File Reference

12.161.1 Detailed Description

Trinomial tree class.

`#include "ql/Lattices/lattice2d.hpp"`

Include dependency graph for lattice2d.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

12.162 ql/Lattices/lattice2d.hpp File Reference

12.162.1 Detailed Description

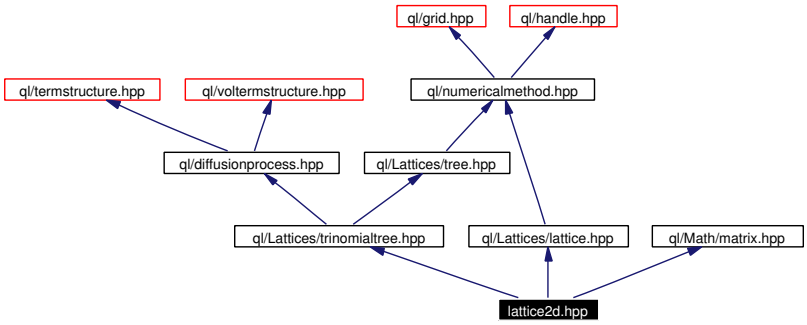
Two-dimensional tree class.

`#include <ql/Lattices/lattice.hpp>`

`#include <ql/Lattices/trinomialtree.hpp>`

`#include <ql/Math/matrix.hpp>`

Include dependency graph for lattice2d.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

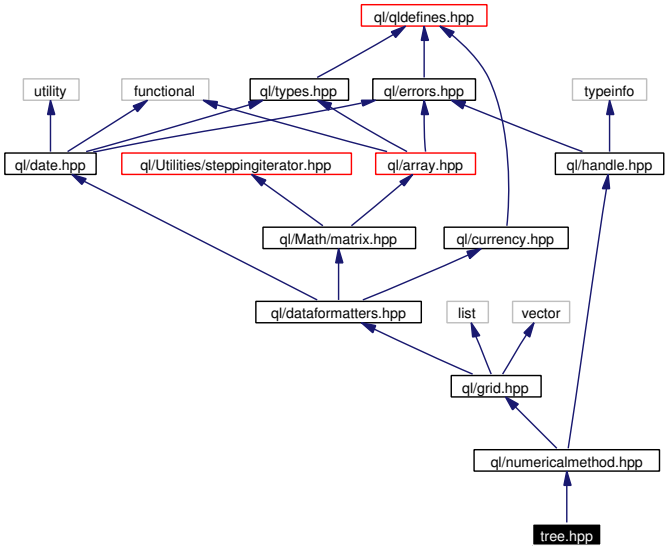
12.163 ql/Lattices/tree.hpp File Reference

12.163.1 Detailed Description

Tree class.

`#include <ql/numericalmethod.hpp>`

Include dependency graph for tree.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

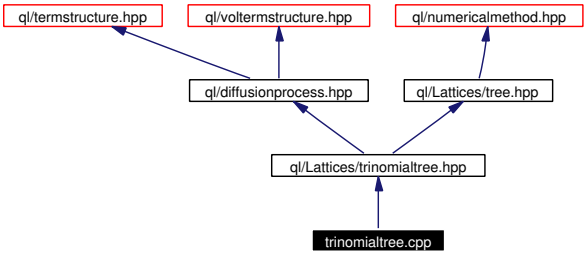
12.164 ql/Lattices/trinomialtree.cpp File Reference

12.164.1 Detailed Description

Trinomial tree class.

`#include "ql/Lattices/trinomialtree.hpp"`

Include dependency graph for trinomialtree.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

12.165 ql/Lattices/trinomialtree.hpp File Reference

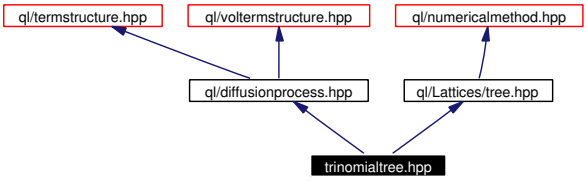
12.165.1 Detailed Description

Trinomial tree class.

`#include <ql/diffusionprocess.hpp>`

`#include <ql/Lattices/tree.hpp>`

Include dependency graph for trinomialtree.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Lattices**

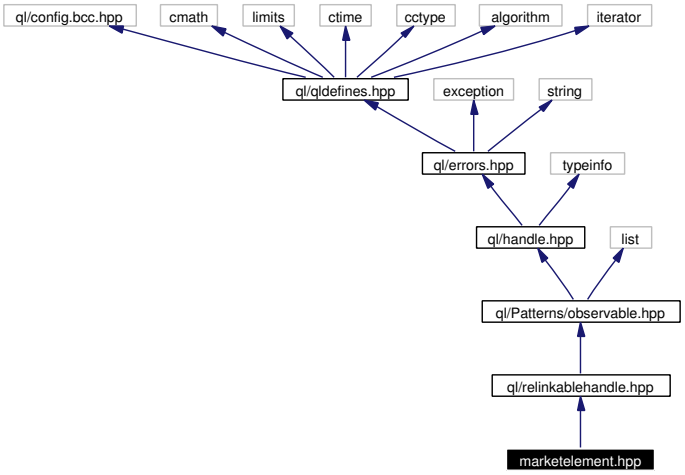
12.166 ql/marketelement.hpp File Reference

12.166.1 Detailed Description

purely virtual base class for market observables

`#include <ql/relinkablehandle.hpp>`

Include dependency graph for marketelement.hpp:



Namespaces

- namespace **QuantLib**

12.167 ql/Math/bicubicsplineinterpolation.hpp File Reference

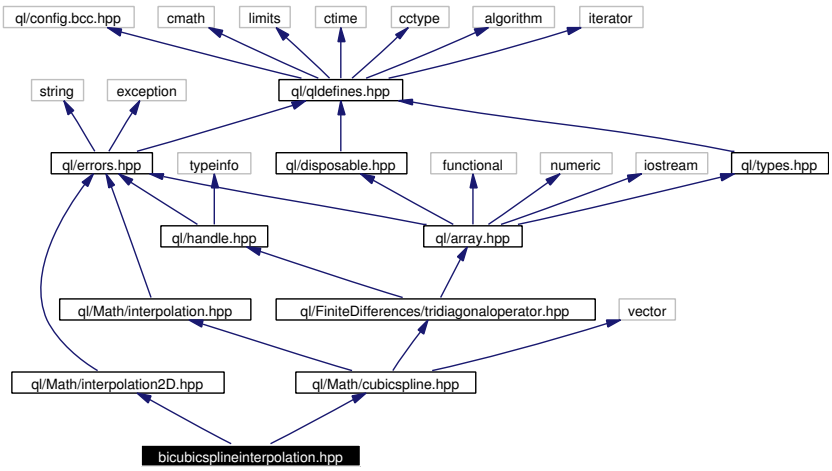
12.167.1 Detailed Description

bicubic spline interpolation between discrete points

`#include <ql/Math/interpolation2D.hpp>`

`#include <ql/Math/cubicspline.hpp>`

Include dependency graph for bicubicsplineinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

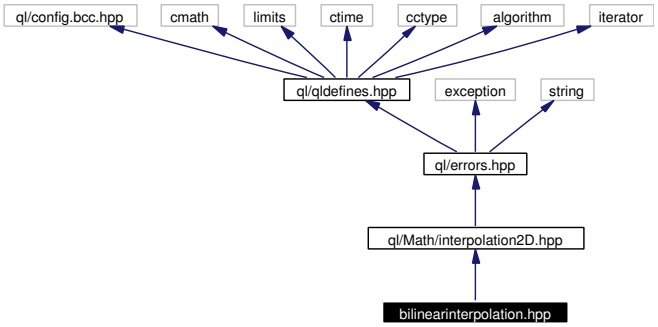
12.168 ql/Math/bilinearinterpolation.hpp File Reference

12.168.1 Detailed Description

bilinear interpolation between discrete points

`#include <ql/Math/interpolation2D.hpp>`

Include dependency graph for bilinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.169 ql/Math/chisquaredistribution.cpp File Reference

12.169.1 Detailed Description

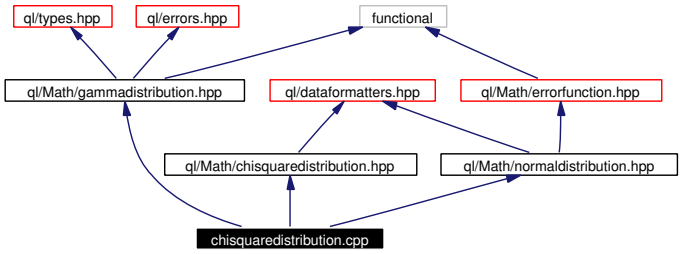
normal, cumulative and inverse cumulative distributions

`#include <ql/Math/chisquaredistribution.hpp>`

`#include <ql/Math/gammadistribution.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for chisquaredistribution.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

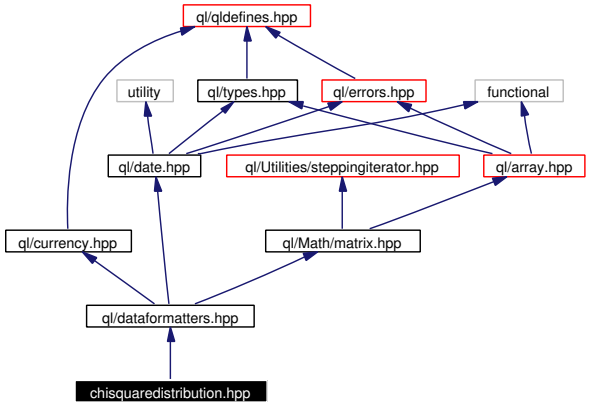
12.170 ql/Math/chisquaredistribution.hpp File Reference

12.170.1 Detailed Description

Chi-square (central and non-central) distributions.

`#include <ql/dataformatters.hpp>`

Include dependency graph for chisquaredistribution.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.171 ql/Math/cubicspline.hpp File Reference

12.171.1 Detailed Description

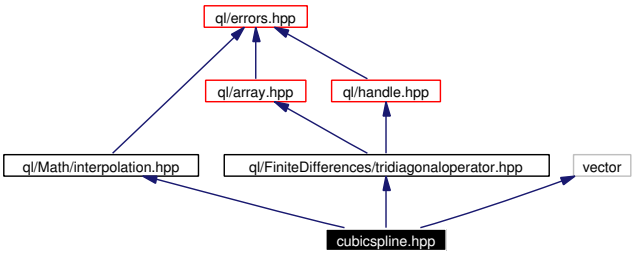
cubic spline interpolation between discrete points

`#include <ql/Math/interpolation.hpp>`

`#include <ql/FiniteDifferences/tridiagonaloperator.hpp>`

`#include <vector>`

Include dependency graph for cubicspline.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

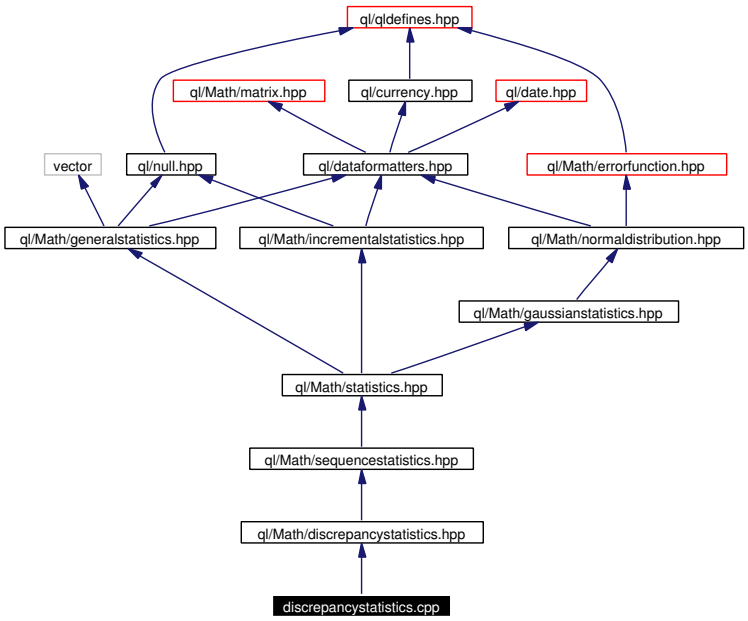
12.172 ql/Math/discrepancystatistics.cpp File Reference

12.172.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

`#include <ql/Math/discrepancystatistics.hpp>`

Include dependency graph for discrepancystatistics.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

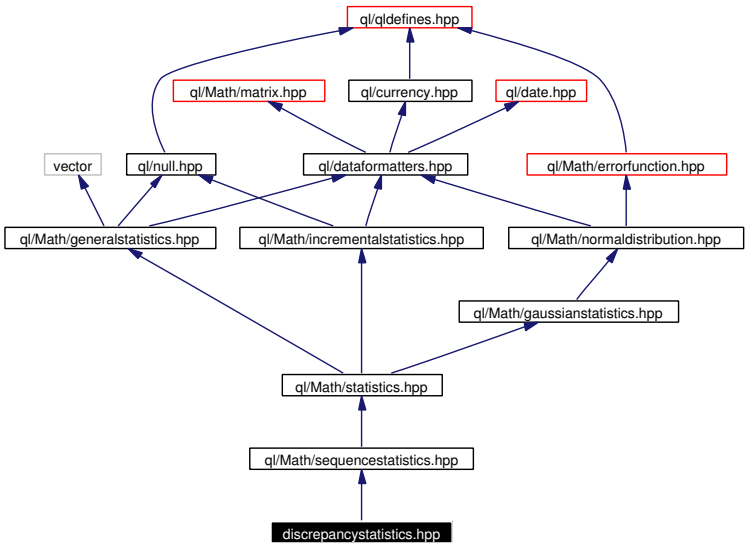
12.173 ql/Math/discrepancystatistics.hpp File Reference

12.173.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

`#include <ql/Math/sequencestatistics.hpp>`

Include dependency graph for discrepancystatistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.174 ql/Math/errorfunction.cpp File Reference

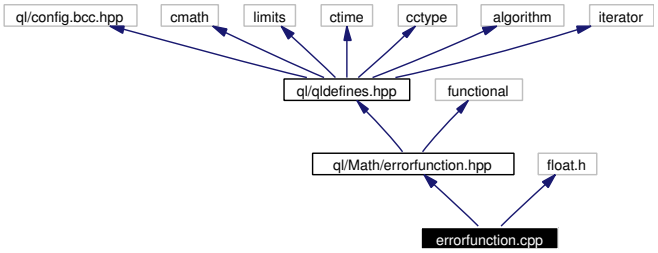
12.174.1 Detailed Description

Error function.

`#include <ql/Math/errorfunction.hpp>`

`#include <float.h>`

Include dependency graph for errorfunction.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.175 ql/Math/errorfunction.hpp File Reference

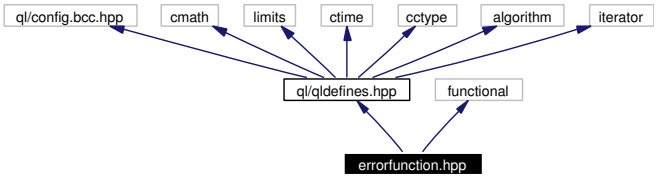
12.175.1 Detailed Description

Error function.

`#include <ql/qldefines.hpp>`

`#include <functional>`

Include dependency graph for errorfunction.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.176 ql/Math/functional.hpp File Reference

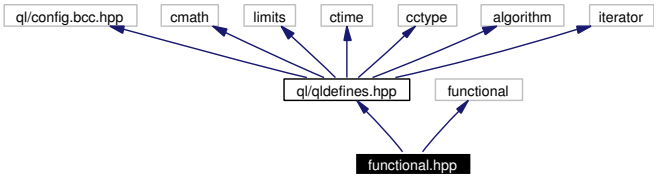
12.176.1 Detailed Description

functionals and combinators not included in the STL

`#include <ql/qldefines.hpp>`

`#include <functional>`

Include dependency graph for functional.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

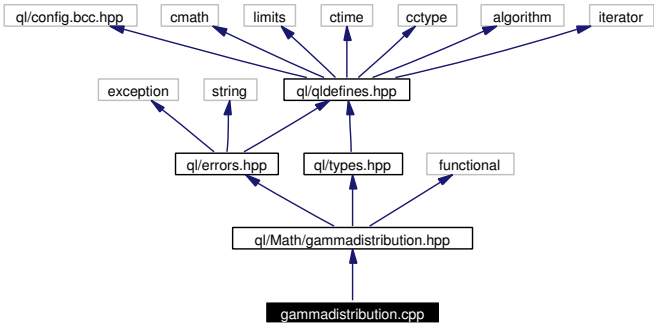
12.177 ql/Math/gammadistribution.cpp File Reference

12.177.1 Detailed Description

Gamma function.

`#include <ql/Math/gammadistribution.hpp>`

Include dependency graph for gammadistribution.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.178 ql/Math/gammadistribution.hpp File Reference

12.178.1 Detailed Description

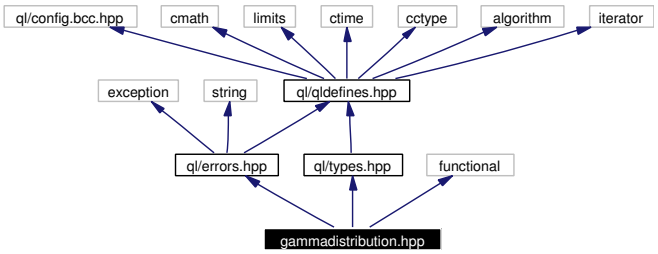
Gamma distribution.

`#include <ql/errors.hpp>`

`#include <ql/types.hpp>`

`#include <functional>`

Include dependency graph for gammadistribution.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

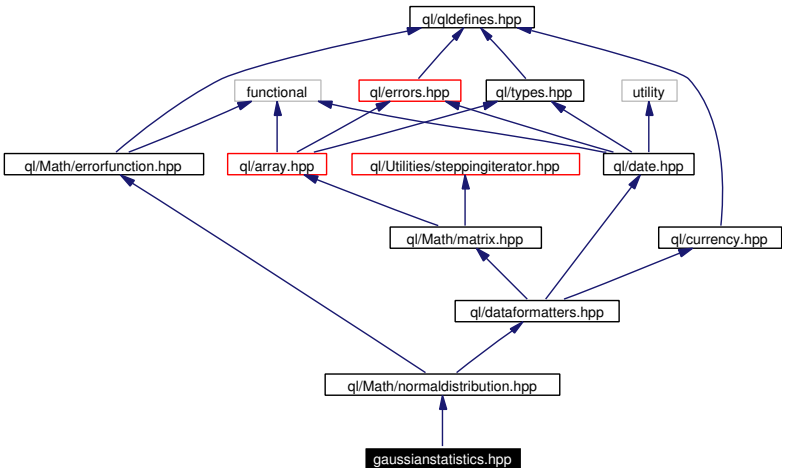
12.179 ql/Math/gaussianstatistics.hpp File Reference

12.179.1 Detailed Description

statistics tool for gaussian-assumption risk measures

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for gaussianstatistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.180 ql/Math/generalstatistics.cpp File Reference

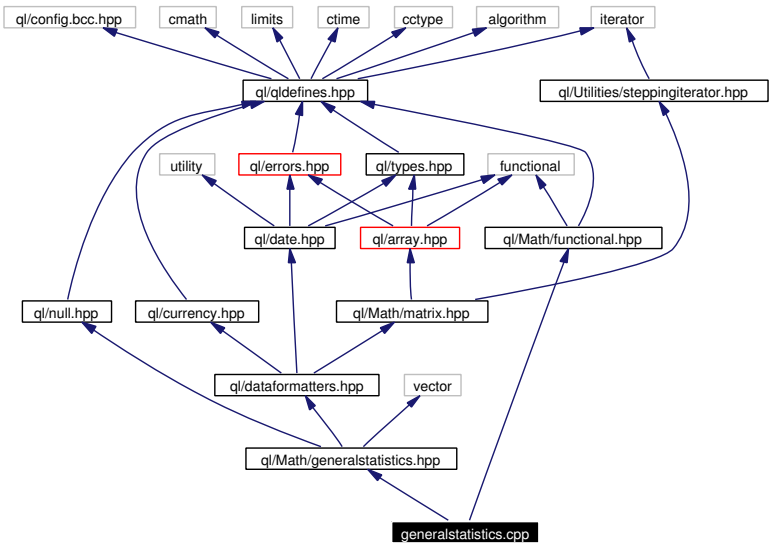
12.180.1 Detailed Description

statistics tool

`#include <ql/Math/generalstatistics.hpp>`

`#include <ql/Math/functional.hpp>`

Include dependency graph for generalstatistics.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.181 ql/Math/generalstatistics.hpp File Reference

12.181.1 Detailed Description

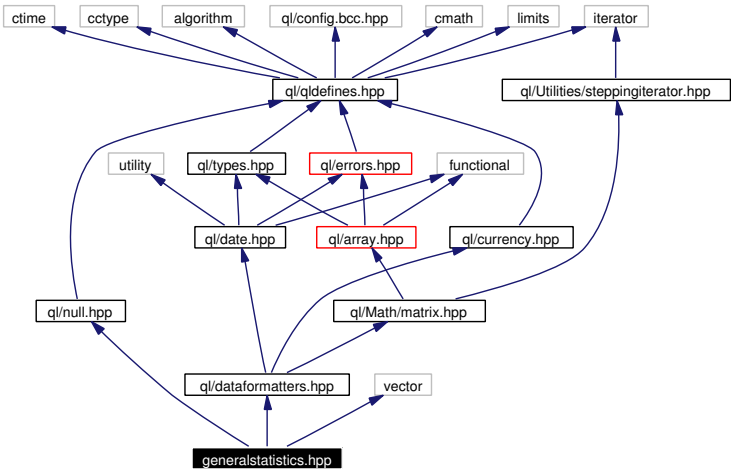
statistics tool

`#include <ql/null.hpp>`

`#include <ql/dataformatters.hpp>`

`#include <vector>`

Include dependency graph for generalstatistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

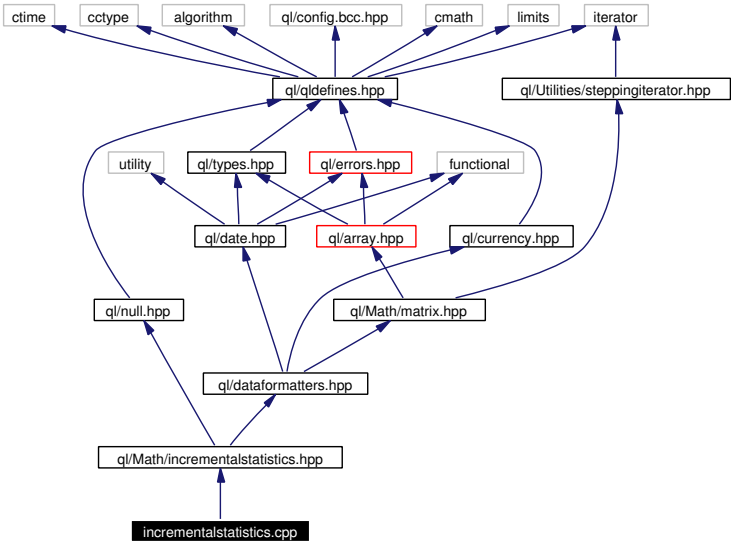
12.182 ql/Math/incrementalstatistics.cpp File Reference

12.182.1 Detailed Description

statistics tool based on incremental accumulation

`#include <ql/Math/incrementalstatistics.hpp>`

Include dependency graph for incrementalstatistics.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.183 ql/Math/incrementalstatistics.hpp File Reference

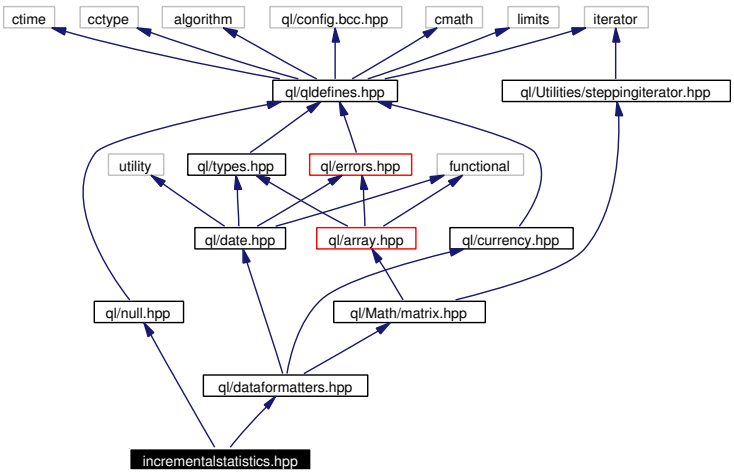
12.183.1 Detailed Description

statistics tool based on incremental accumulation

`#include <ql/null.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for incrementalstatistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

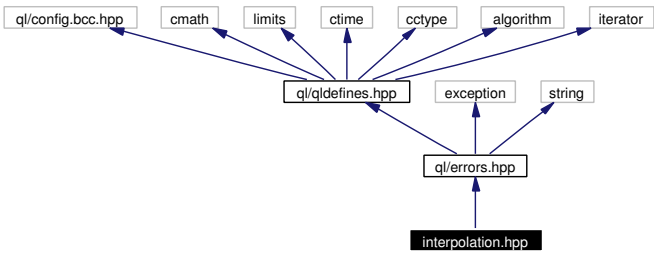
12.184 ql/Math/interpolation.hpp File Reference

12.184.1 Detailed Description

abstract base classes for interpolations

`#include <ql/errors.hpp>`

Include dependency graph for interpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

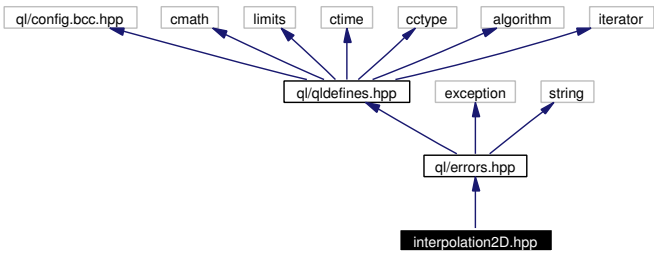
12.185 ql/Math/interpolation2D.hpp File Reference

12.185.1 Detailed Description

abstract base classes for 2-D interpolations

`#include <ql/errors.hpp>`

Include dependency graph for interpolation2D.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.186 ql/Math/interpolationtraits.hpp File Reference

12.186.1 Detailed Description

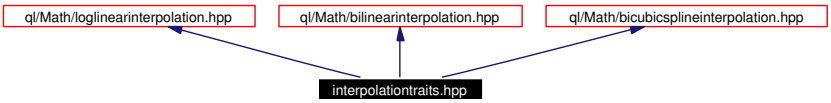
traits classes for interpolation algorithms

`#include <ql/Math/loglinearinterpolation.hpp>`

`#include <ql/Math/bilinearinterpolation.hpp>`

`#include <ql/Math/bicubicsplineinterpolation.hpp>`

Include dependency graph for interpolationtraits.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.187 ql/Math/kronrodintegral.hpp File Reference

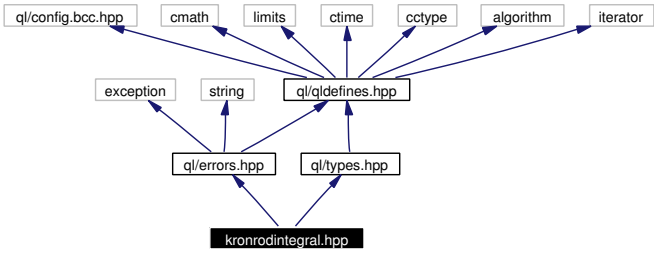
12.187.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

`#include <ql/types.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for kronrodintegral.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

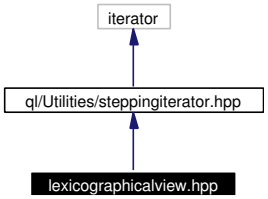
12.188 ql/Math/lexicographicalview.hpp File Reference

12.188.1 Detailed Description

Lexicographical 2-D view of a contiguous set of data.

`#include <ql/Utilities/steppingiterator.hpp>`

Include dependency graph for lexicographicalview.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

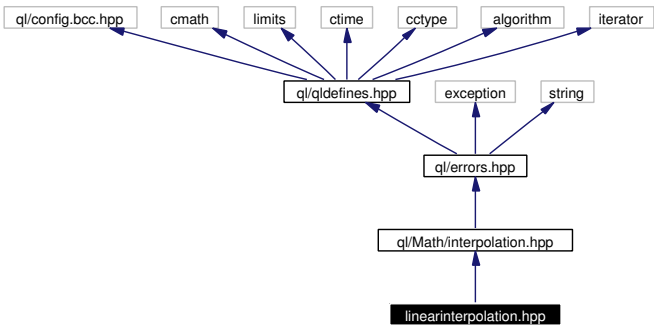
12.189 ql/Math/linearinterpolation.hpp File Reference

12.189.1 Detailed Description

linear interpolation between discrete points

`#include <ql/Math/interpolation.hpp>`

Include dependency graph for linearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

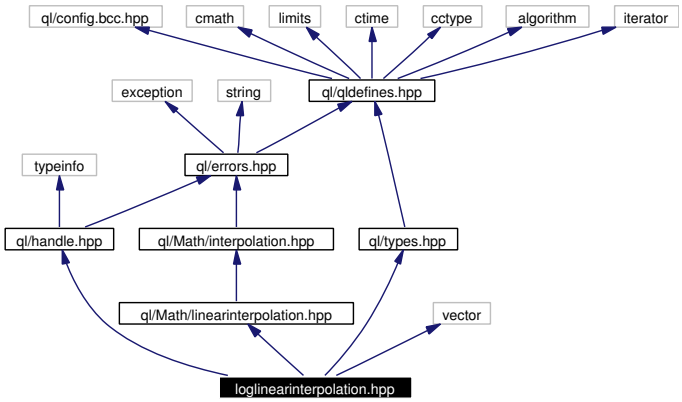
12.190 ql/Math/loglinearinterpolation.hpp File Reference

12.190.1 Detailed Description

log-linear interpolation between discrete points

```
#include <ql/types.hpp>
#include <ql/handle.hpp>
#include <ql/Math/linearinterpolation.hpp>
#include <vector>
```

Include dependency graph for loglinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.191 ql/Math/matrix.cpp File Reference

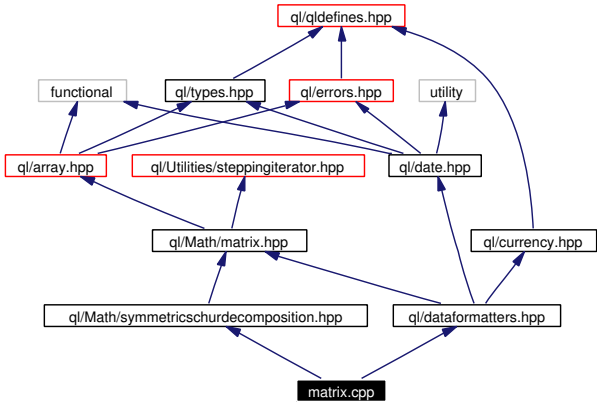
12.191.1 Detailed Description

matrix used in linear algebra.

`#include <ql/Math/symmetricschurdecomposition.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for matrix.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.192 ql/Math/matrix.hpp File Reference

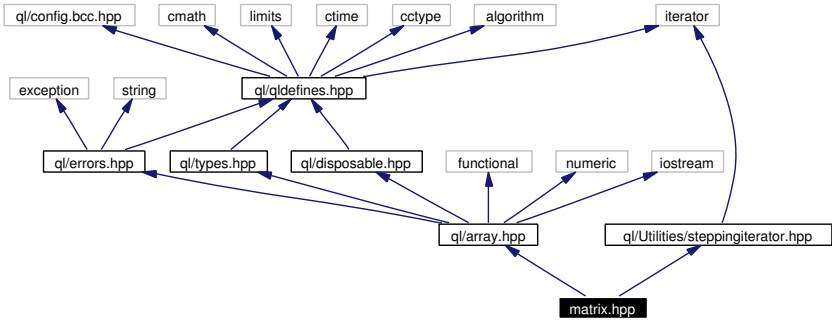
12.192.1 Detailed Description

matrix used in linear algebra.

`#include <ql/array.hpp>`

`#include <ql/Utilities/steppingiterator.hpp>`

Include dependency graph for matrix.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

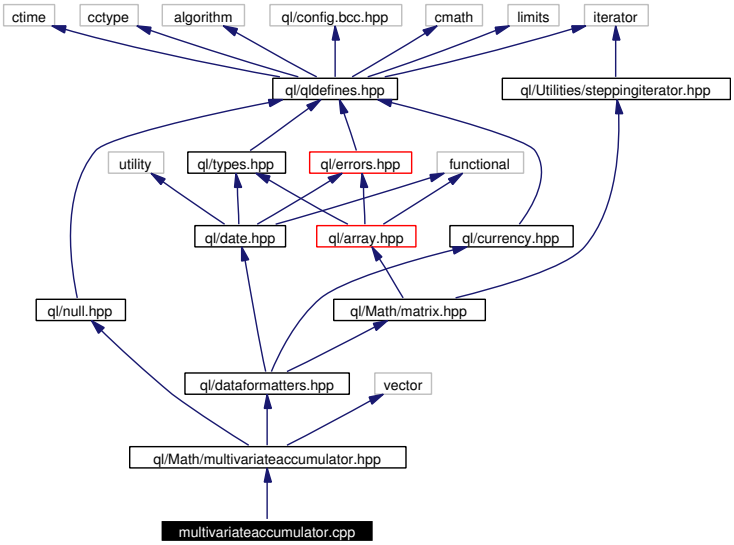
12.193 ql/Math/multivariateaccumulator.cpp File Reference

12.193.1 Detailed Description

A simple accumulator for vector-type samples.

`#include <ql/Math/multivariateaccumulator.hpp>`

Include dependency graph for multivariateaccumulator.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.194 ql/Math/multivariateaccumulator.hpp File Reference

12.194.1 Detailed Description

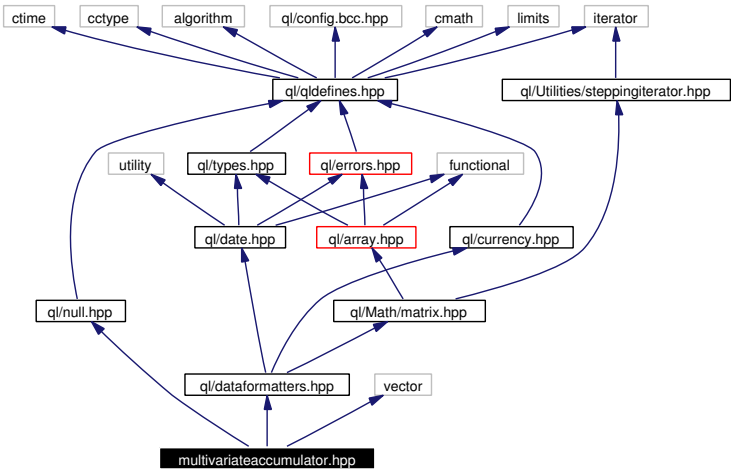
A simple accumulator for vector-type samples.

`#include <ql/null.hpp>`

`#include <ql/dataformatters.hpp>`

`#include <vector>`

Include dependency graph for multivariateaccumulator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

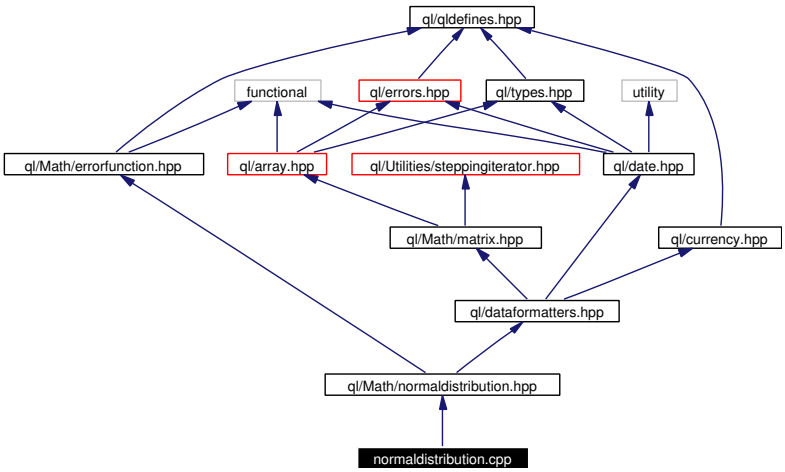
12.195 ql/Math/normaldistribution.cpp File Reference

12.195.1 Detailed Description

normal, cumulative and inverse cumulative distributions

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for normaldistribution.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.196 ql/Math/normaldistribution.hpp File Reference

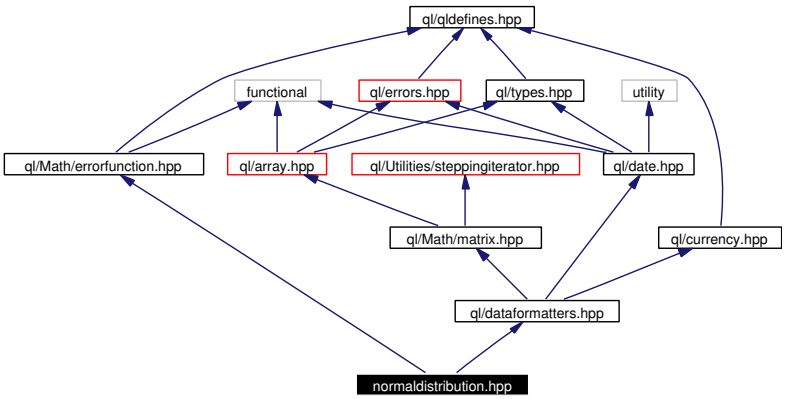
12.196.1 Detailed Description

normal, cumulative and inverse cumulative distributions

`#include <ql/dataformatters.hpp>`

`#include <ql/Math/errorfunction.hpp>`

Include dependency graph for normaldistribution.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

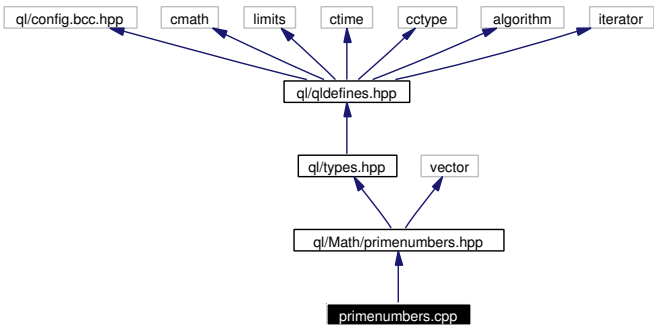
12.197 ql/Math/primenumbers.cpp File Reference

12.197.1 Detailed Description

Prime numbers calculator.

`#include "ql/Math/primenumbers.hpp"`

Include dependency graph for primenumbers.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.198 ql/Math/primenumbers.hpp File Reference

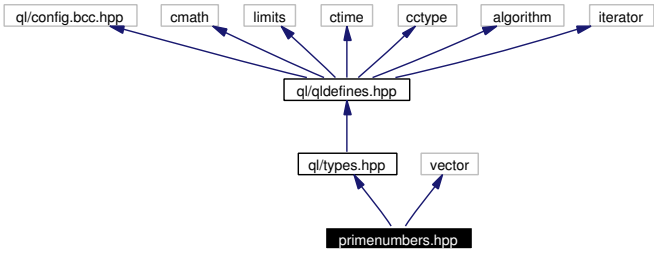
12.198.1 Detailed Description

Prime numbers calculator.

`#include <ql/types.hpp>`

`#include <vector>`

Include dependency graph for primenumbers.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.199 ql/Math/riskmeasures.hpp File Reference

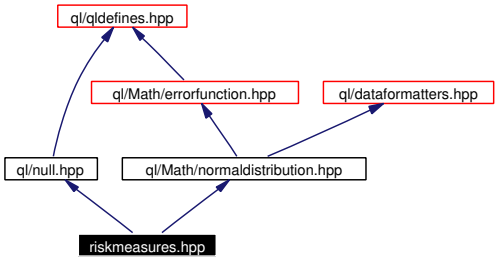
12.199.1 Detailed Description

Risk functions.

`#include <ql/null.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for riskmeasures.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.200 ql/Math/segmentintegral.hpp File Reference

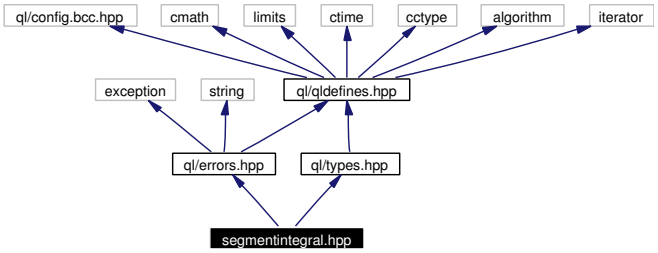
12.200.1 Detailed Description

Integral of a one-dimensional function.

`#include <ql/types.hpp>`

`#include <ql/errors.hpp>`

Include dependency graph for segmentintegral.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

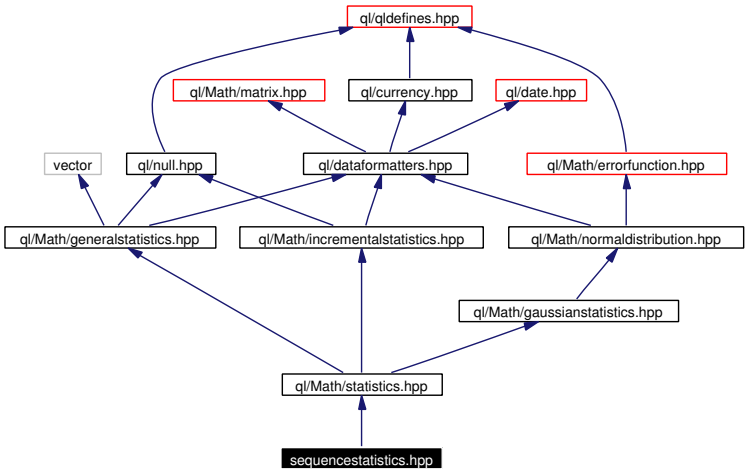
12.201 ql/Math/sequencestatistics.hpp File Reference

12.201.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

`#include <ql/Math/statistics.hpp>`

Include dependency graph for sequencestatistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

Defines

- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID**(METHOD)
- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE**(METHOD)

12.201.2 Define Documentation

12.201.2.1 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)

Value:

```
template <class Stat> \
        std::vector<double> \
        SequenceStatistics<Stat>::METHOD() const { \
            for (Size i=0; i<dimension_; i++) \
                results_[i] = stats_[i].METHOD(); \
            return results_; \
        }
```

12.201.2.2 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)

Value:

```
template <class Stat> \
        std::vector<double> \
        SequenceStatistics<Stat>::METHOD(double x) const { \
            for (Size i=0; i<dimension_; i++) \
                results_[i] = stats_[i].METHOD(x); \
            return results_; \
        }
```

12.202 ql/Math/statistics.hpp File Reference

12.202.1 Detailed Description

statistics tool with risk measures

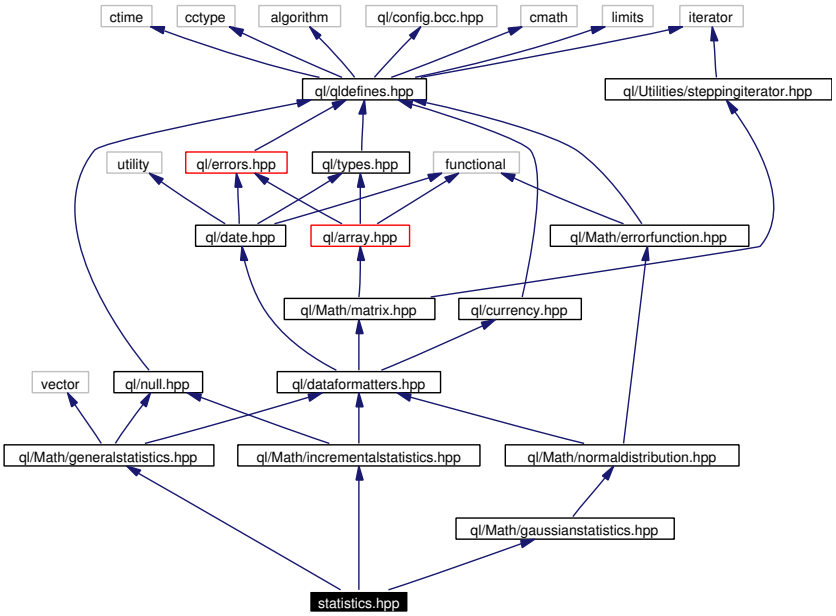
`#include <ql/Math/generalstatistics.hpp>`

`#include <ql/Math/incrementalstatistics.hpp>`

`#include <ql/Math/gaussianstatistics.hpp>`

`#include <ql/Math/riskstatistics.hpp>`

Include dependency graph for statistics.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.203 ql/Math/svd.cpp File Reference

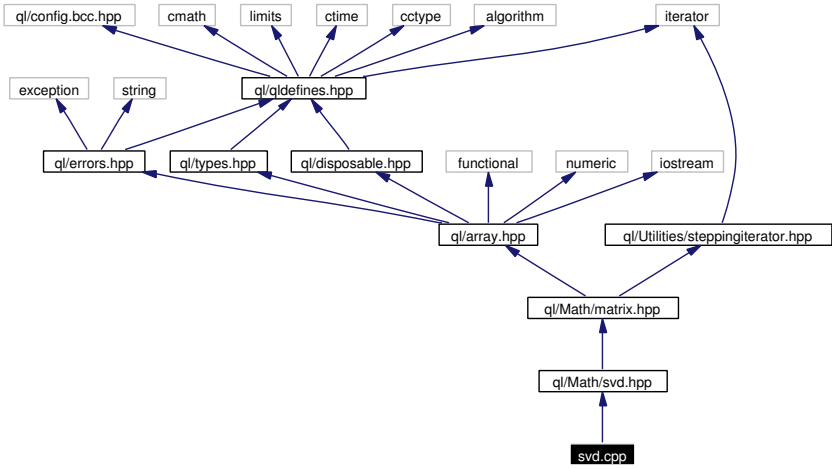
12.203.1 Detailed Description

Calculates Singular Value Decomposition.

in MATLAB notation A = U∗S∗V′

`#include <ql/Math/svd.hpp>`

Include dependency graph for svd.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.204 ql/Math/svd.hpp File Reference

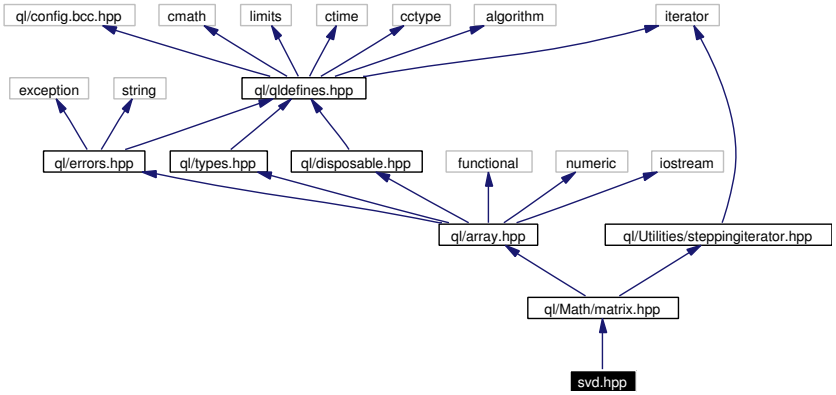
12.204.1 Detailed Description

Calculates Singular Value Decomposition.

in MATLAB notation A = U∗S∗V′

`#include <ql/Math/matrix.hpp>`

Include dependency graph for svd.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

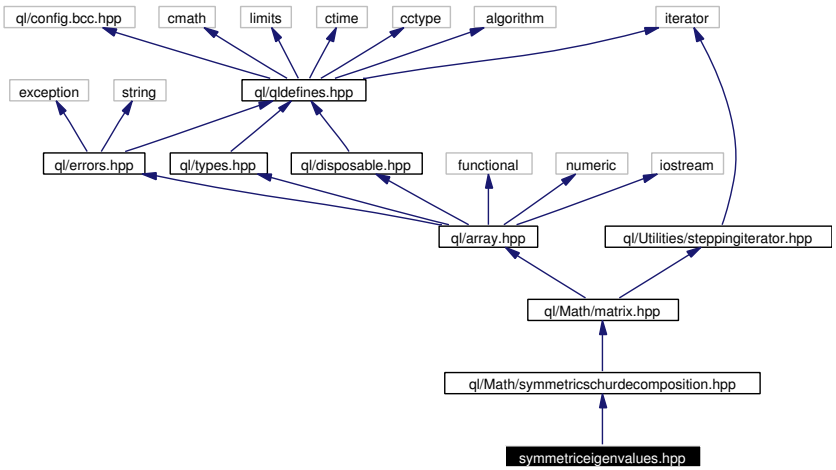
12.205 ql/Math/symmetriceigenvalues.hpp File Reference

12.205.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

`#include <ql/Math/symmetricschurdecomposition.hpp>`

Include dependency graph for symmetriceigenvalues.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.206 ql/Math/symmetricschurdecomposition.cpp File Reference

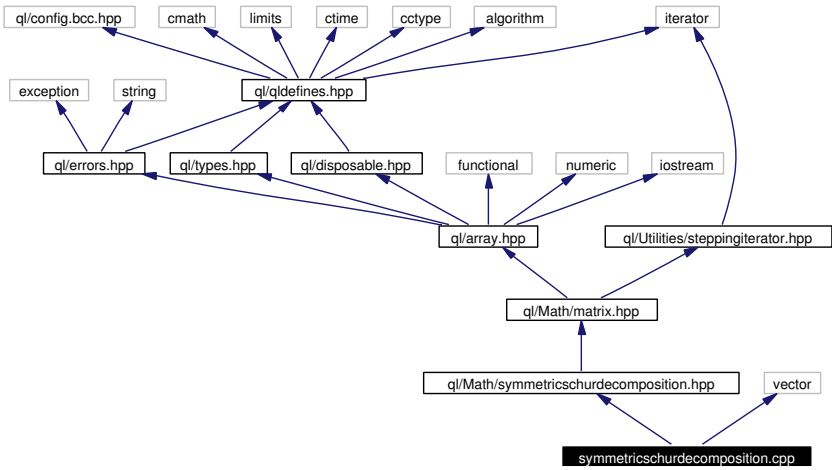
12.206.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

`#include <ql/Math/symmetricschurdecomposition.hpp>`

`#include <vector>`

Include dependency graph for symmetricschurdecomposition.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

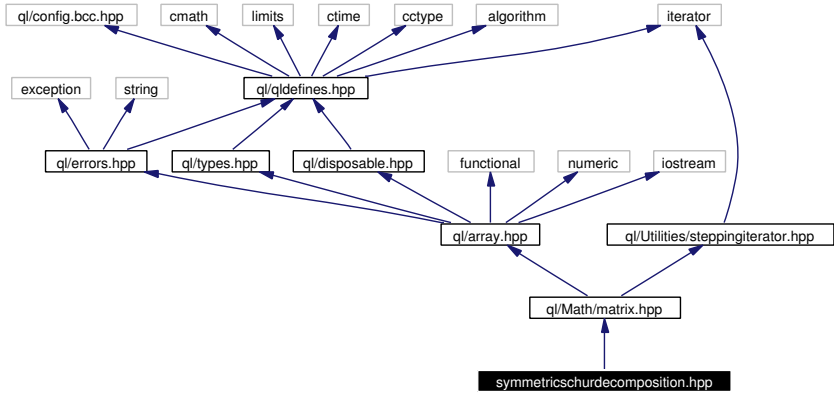
12.207 ql/Math/symmetricschurdecomposition.hpp File Reference

12.207.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

`#include <ql/Math/matrix.hpp>`

Include dependency graph for symmetricschurdecomposition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.208 ql/Math/trapezoidintegral.hpp File Reference

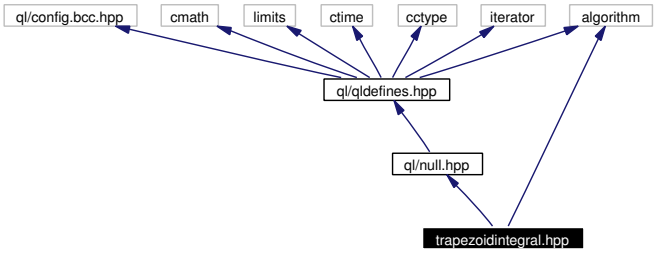
12.208.1 Detailed Description

integral of a one-dimensional function

`#include <ql/null.hpp>`

`#include <algorithm>`

Include dependency graph for trapezoidintegral.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Math**

12.209 ql/MonteCarlo/arithmeticapopathpricer.cpp File Reference

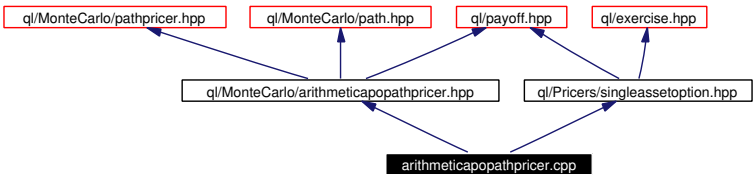
12.209.1 Detailed Description

arithmetic average price option path pricer

`#include <ql/MonteCarlo/arithmeticapopathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for arithmeticapopathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.210 ql/MonteCarlo/arithmeticapopathpricer.hpp File Reference

12.210.1 Detailed Description

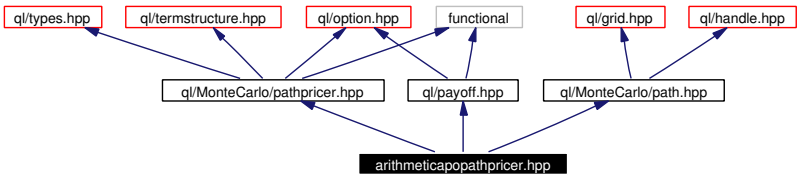
arithmetic average price option path pricer

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for arithmeticapopathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.211 ql/MonteCarlo/arithmeticasopathpricer.cpp File Reference

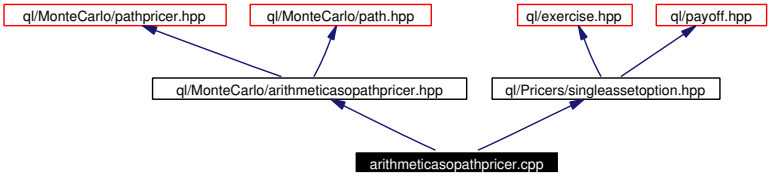
12.211.1 Detailed Description

arithmetic average strike option path pricer

`#include <ql/MonteCarlo/arithmeticasopathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for arithmeticasopathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.212 ql/MonteCarlo/arithmeticasopathpricer.hpp File Reference

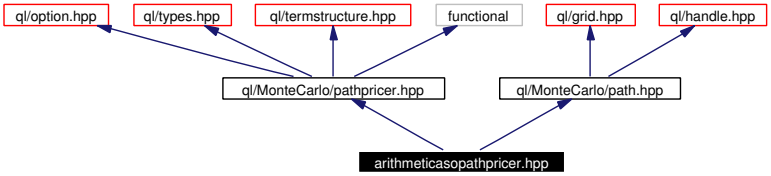
12.212.1 Detailed Description

arithmetic average strike option path pricer

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

Include dependency graph for arithmeticasopathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

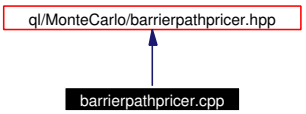
12.213 ql/MonteCarlo/barrierpathpricer.cpp File Reference

12.213.1 Detailed Description

path pricer for Barrier options

`#include <ql/MonteCarlo/barrierpathpricer.hpp>`

Include dependency graph for barrierpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

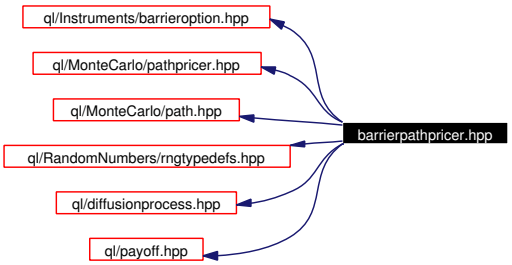
12.214 ql/MonteCarlo/barrierpathpricer.hpp File Reference

12.214.1 Detailed Description

path pricer for Barrier options

```
#include <ql/Instruments/barrieroption.hpp>
#include <ql/MonteCarlo/pathpricer.hpp>
#include <ql/MonteCarlo/path.hpp>
#include <ql/RandomNumbers/rngtypedefs.hpp>
#include <ql/diffusionprocess.hpp>
#include <ql/payoff.hpp>
```

Include dependency graph for barrierpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.215 ql/MonteCarlo/basketpathpricer.cpp File Reference

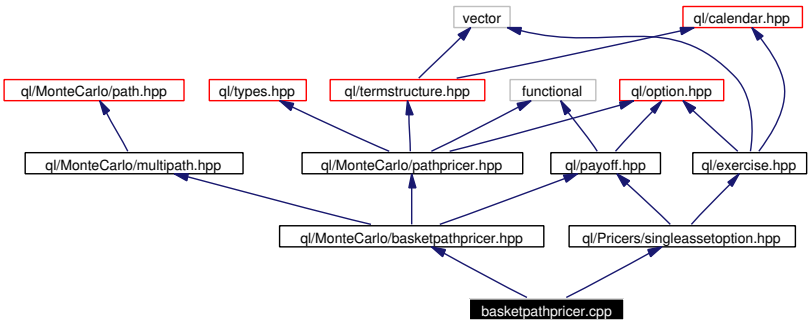
12.215.1 Detailed Description

multipath pricer for European-type basket option

`#include <ql/MonteCarlo/basketpathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for basketpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.216 ql/MonteCarlo/basketpathpricer.hpp File Reference

12.216.1 Detailed Description

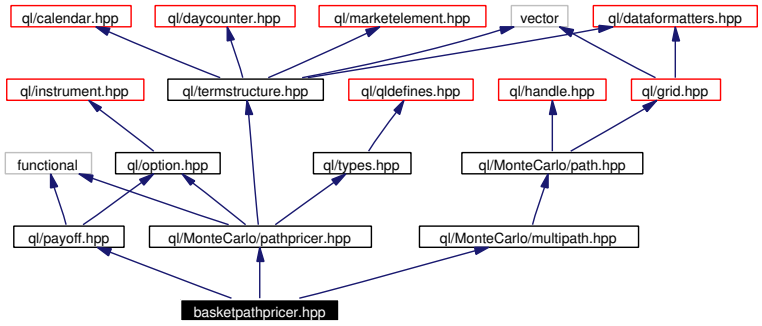
multipath pricer for European-type basket option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for basketpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

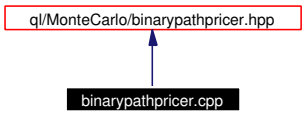
12.217 ql/MonteCarlo/binarypathpricer.cpp File Reference

12.217.1 Detailed Description

path pricer for Binary options

`#include <ql/MonteCarlo/binarypathpricer.hpp>`

Include dependency graph for binarypathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

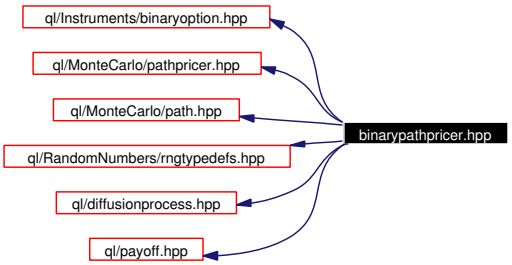
12.218 ql/MonteCarlo/binarypathpricer.hpp File Reference

12.218.1 Detailed Description

path pricer for Binary options

```
#include <ql/Instruments/binaryoption.hpp>
#include <ql/MonteCarlo/pathpricer.hpp>
#include <ql/MonteCarlo/path.hpp>
#include <ql/RandomNumbers/rngtypedefs.hpp>
#include <ql/diffusionprocess.hpp>
#include <ql/payoff.hpp>
```

Include dependency graph for binarypathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.219 ql/MonteCarlo/brownianbridge.hpp File Reference

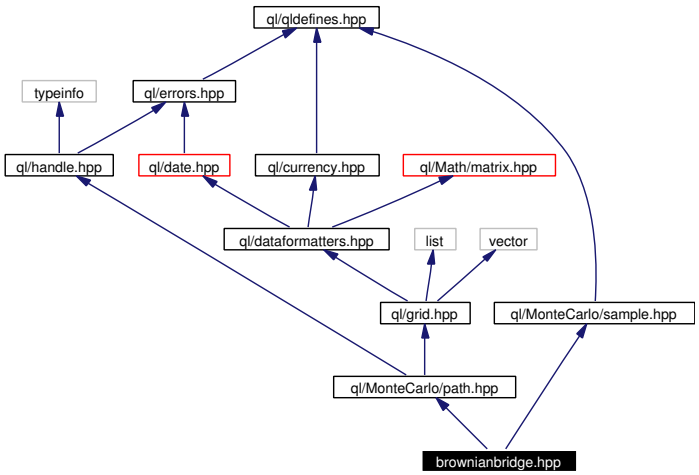
12.219.1 Detailed Description

Browian bridge.

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for brownianbridge.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.220 ql/MonteCarlo/cliquetoptionpathpricer.cpp File Reference

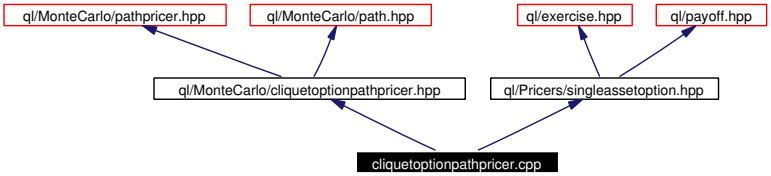
12.220.1 Detailed Description

path pricer for cliquet options

`#include <ql/MonteCarlo/cliquetoptionpathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for cliquetoptionpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.221 ql/MonteCarlo/cliquetoptionpathpricer.hpp File Reference

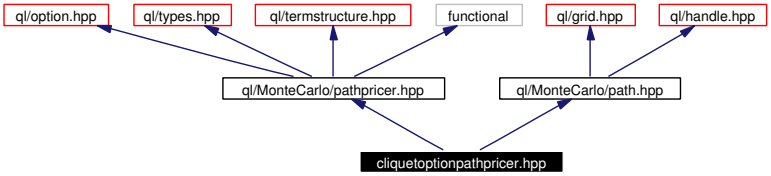
12.221.1 Detailed Description

path pricer for cliquet options

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

Include dependency graph for cliquetoptionpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.222 ql/MonteCarlo/europeanpathpricer.cpp File Reference

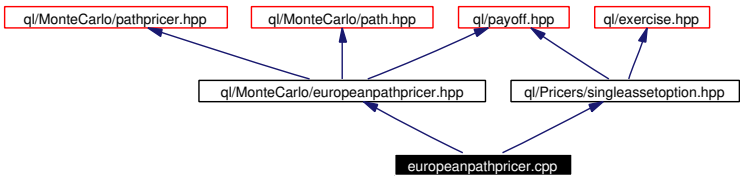
12.222.1 Detailed Description

path pricer for European options

`#include <ql/MonteCarlo/europeanpathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for europeanpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.223 ql/MonteCarlo/europeanpathpricer.hpp File Reference

12.223.1 Detailed Description

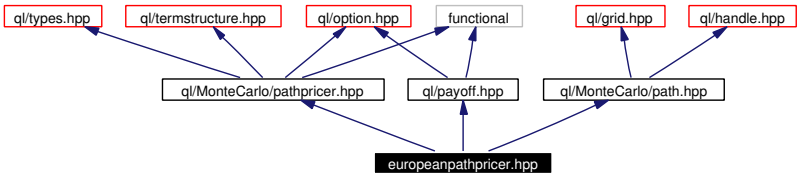
path pricer for European options

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for europeanpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

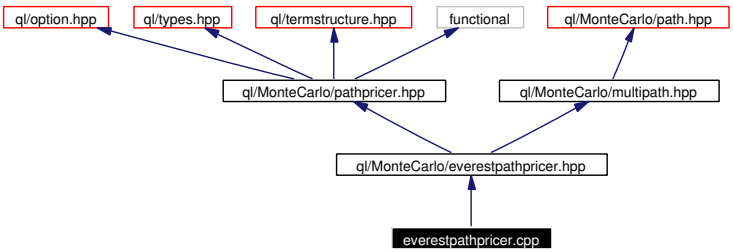
12.224 ql/MonteCarlo/everestpathpricer.cpp File Reference

12.224.1 Detailed Description

path pricer for European-type Everest option

`#include <ql/MonteCarlo/everestpathpricer.hpp>`

Include dependency graph for everestpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.225 ql/MonteCarlo/everestpathpricer.hpp File Reference

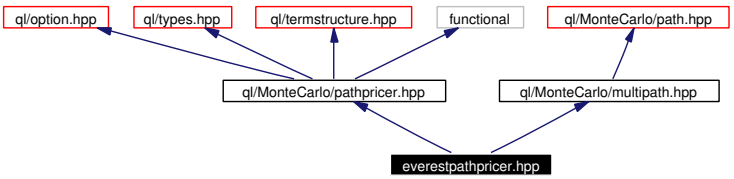
12.225.1 Detailed Description

path pricer for European-type Everest option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

Include dependency graph for everestpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.226 ql/MonteCarlo/geometricapopathpricer.cpp File Reference

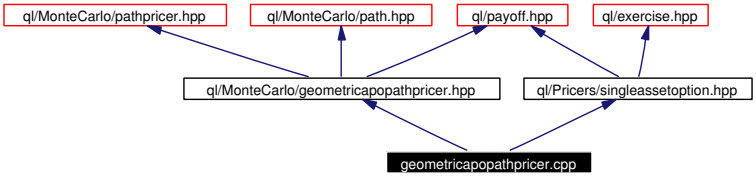
12.226.1 Detailed Description

path pricer for geometric average price option

`#include <ql/MonteCarlo/geometricapopathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for geometricapopathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.227 ql/MonteCarlo/geometricapopathpricer.hpp File Reference

12.227.1 Detailed Description

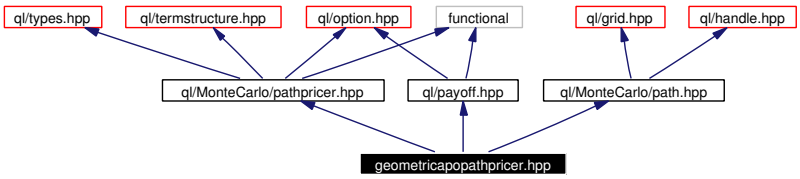
path pricer for geometric average price option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for geometricapopathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.228 ql/MonteCarlo/geometricasopathpricer.cpp File Reference

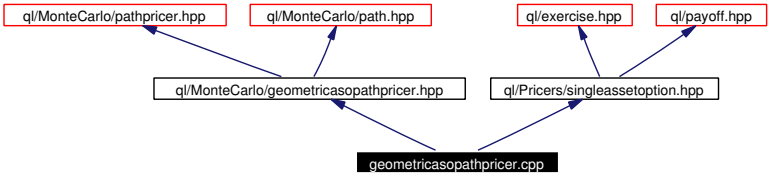
12.228.1 Detailed Description

path pricer for geometric average strike option

`#include <ql/MonteCarlo/geometricasopathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for geometricasopathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.229 ql/MonteCarlo/geometricasopathpricer.hpp File Reference

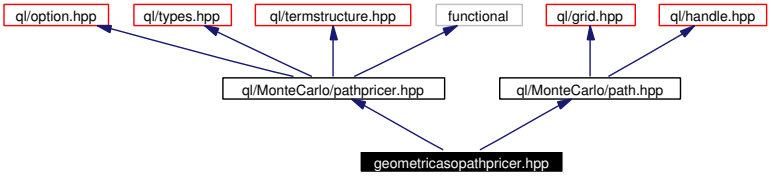
12.229.1 Detailed Description

path pricer for geometric average strike option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

Include dependency graph for geometricasopathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

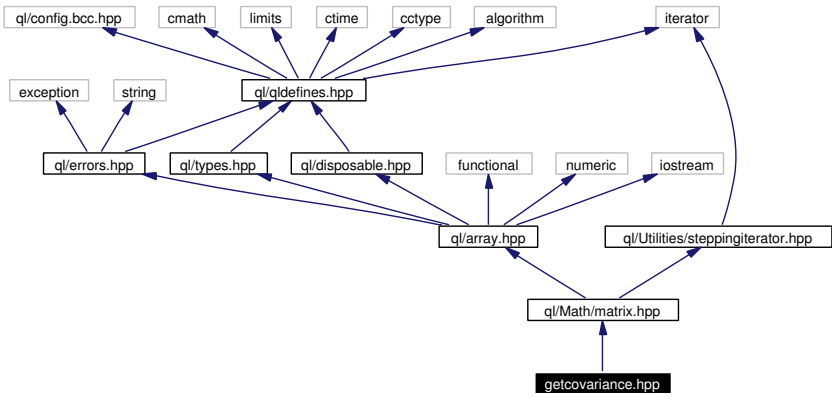
12.230 ql/MonteCarlo/getcovariance.hpp File Reference

12.230.1 Detailed Description

Covariance matrix calculation.

`#include <ql/Math/matrix.hpp>`

Include dependency graph for getcovariance.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

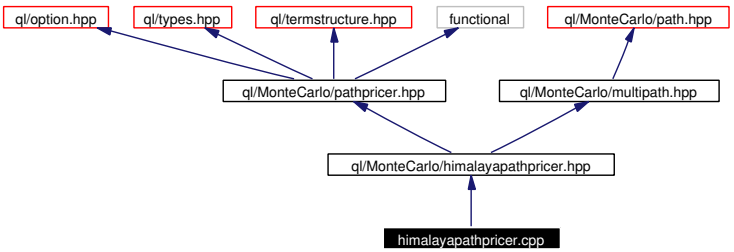
12.231 ql/MonteCarlo/himalayapathpricer.cpp File Reference

12.231.1 Detailed Description

multipath pricer for European-type Himalaya option

`#include <ql/MonteCarlo/himalayapathpricer.hpp>`

Include dependency graph for himalayapathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.232 ql/MonteCarlo/himalayapathpricer.hpp File Reference

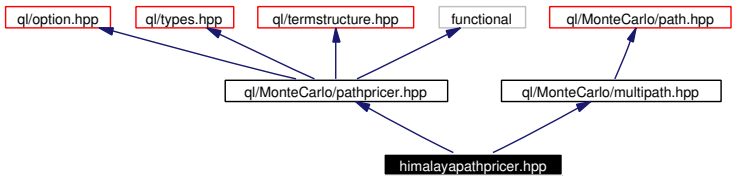
12.232.1 Detailed Description

multipath pricer for European-type Himalaya option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

Include dependency graph for himalayapathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

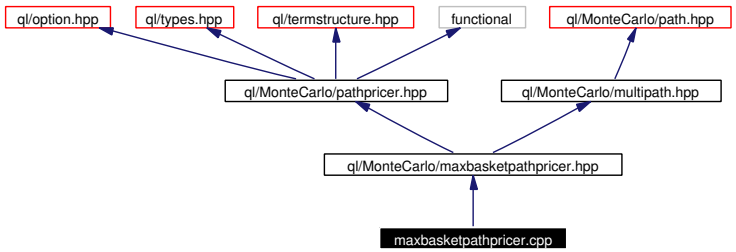
12.233 ql/MonteCarlo/maxbasketpathpricer.cpp File Reference

12.233.1 Detailed Description

multipath pricer for max basket option

`#include <ql/MonteCarlo/maxbasketpathpricer.hpp>`

Include dependency graph for maxbasketpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.234 ql/MonteCarlo/maxbasketpathpricer.hpp File Reference

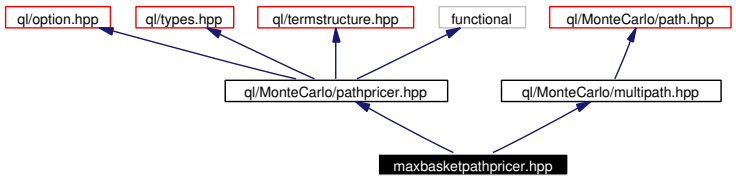
12.234.1 Detailed Description

multipath pricer for max basket option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

Include dependency graph for maxbasketpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

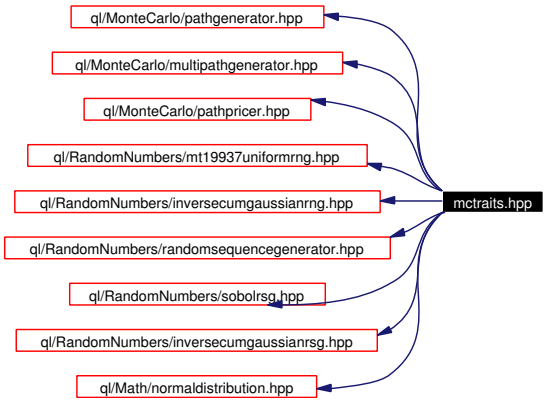
12.235 ql/MonteCarlo/mctraits.hpp File Reference

12.235.1 Detailed Description

Monte Carlo policy descriptors.

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/MonteCarlo/multipathgenerator.hpp>
#include <ql/MonteCarlo/pathpricer.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumgaussianrng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumgaussianrsg.hpp>
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for mctraits.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.236 ql/MonteCarlo/mctypedefs.hpp File Reference

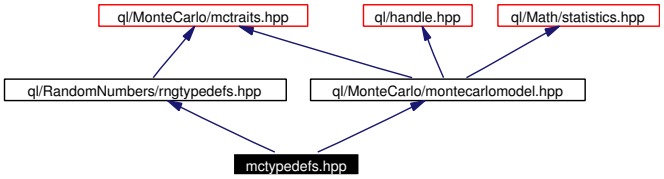
12.236.1 Detailed Description

Default choices for template instantiations.

`#include <ql/RandomNumbers/rngtypedefs.hpp>`

`#include <ql/MonteCarlo/montecarlomodel.hpp>`

Include dependency graph for mctypedefs.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

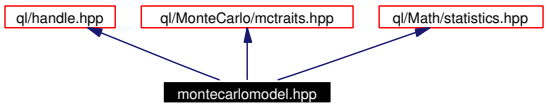
12.237 ql/MonteCarlo/montecarlomodel.hpp File Reference

12.237.1 Detailed Description

General purpose Monte Carlo model.

```
#include <ql/handle.hpp>
#include <ql/MonteCarlo/mctraits.hpp>
#include <ql/Math/statistics.hpp>
```

Include dependency graph for montecarlomodel.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

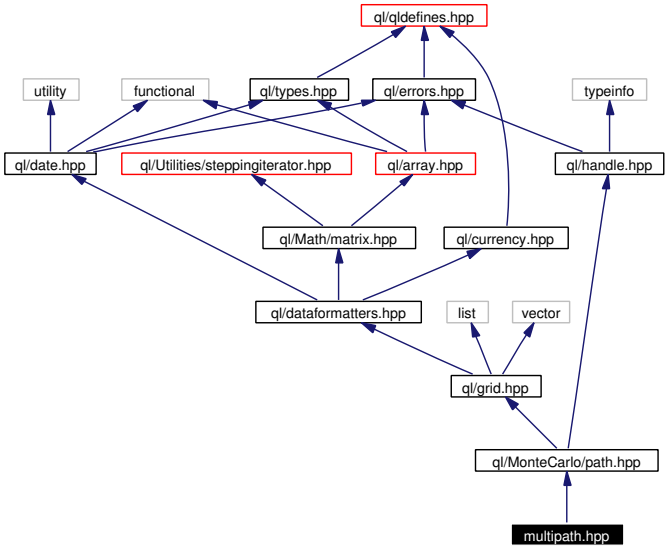
12.238 ql/MonteCarlo/multipath.hpp File Reference

12.238.1 Detailed Description

Correlated multiple asset paths.

`#include <ql/MonteCarlo/path.hpp>`

Include dependency graph for multipath.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.239 ql/MonteCarlo/multipathgenerator.hpp File Reference

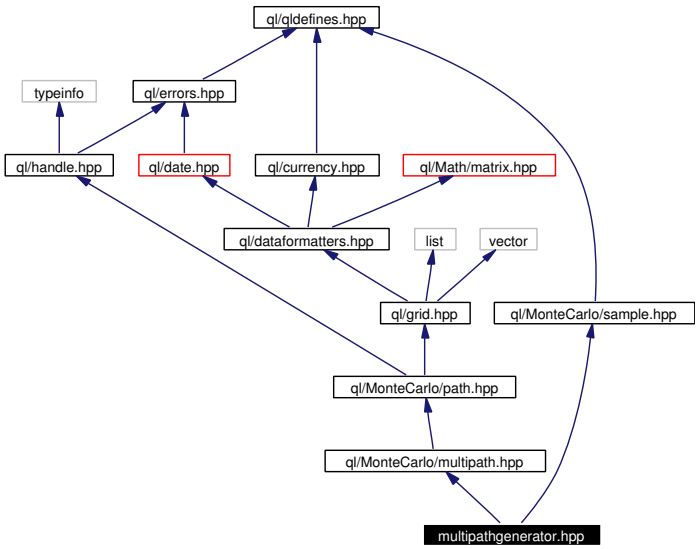
12.239.1 Detailed Description

Generates a multi path from a random-array generator.

`#include <ql/MonteCarlo/multipath.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for multipathgenerator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

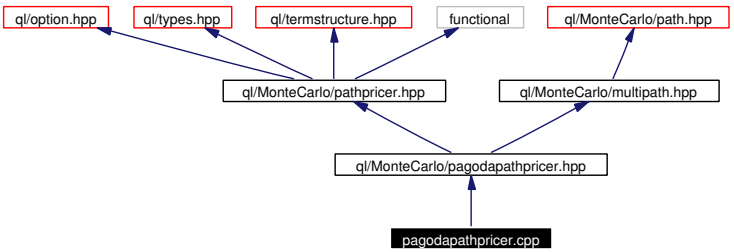
12.240 ql/MonteCarlo/pagodapathpricer.cpp File Reference

12.240.1 Detailed Description

path pricer for pagoda options

`#include <ql/MonteCarlo/pagodapathpricer.hpp>`

Include dependency graph for pagodapathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.241 ql/MonteCarlo/pagodapathpricer.hpp File Reference

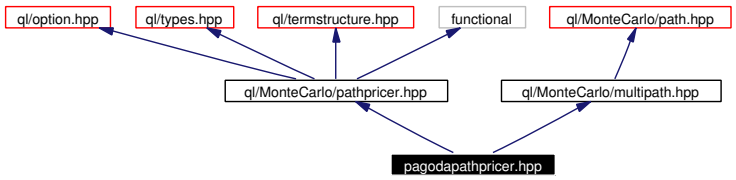
12.241.1 Detailed Description

path pricer for pagoda options

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/multipath.hpp>`

Include dependency graph for pagodapathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.242 ql/MonteCarlo/path.hpp File Reference

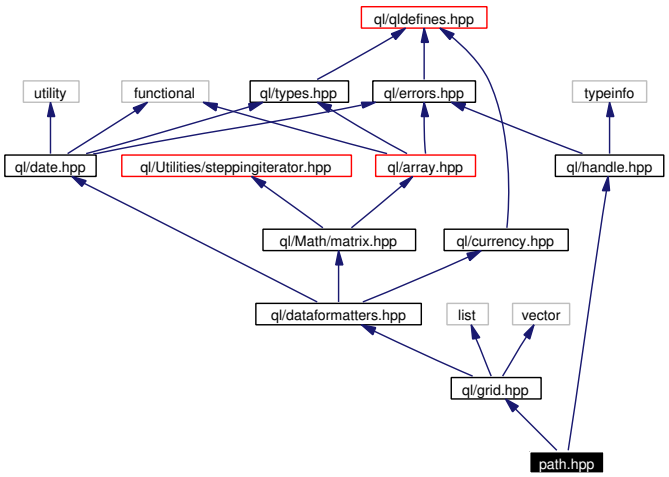
12.242.1 Detailed Description

single factor random walk

`#include <ql/grid.hpp>`

`#include <ql/handle.hpp>`

Include dependency graph for path.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.243 ql/MonteCarlo/pathgenerator.hpp File Reference

12.243.1 Detailed Description

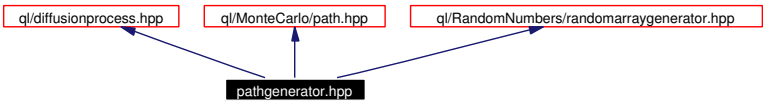
Generates random paths using a sequence generator.

`#include <ql/diffusionprocess.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/RandomNumbers/randomarraygenerator.hpp>`

Include dependency graph for pathgenerator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.244 ql/MonteCarlo/pathpricer.hpp File Reference

12.244.1 Detailed Description

base class for single-path pricers

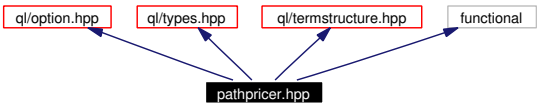
`#include <ql/option.hpp>`

`#include <ql/types.hpp>`

`#include <ql/termstructure.hpp>`

`#include <functional>`

Include dependency graph for pathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.245 ql/MonteCarlo/performanceoptionpathpricer.cpp File Reference

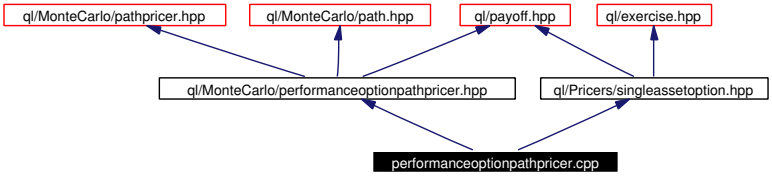
12.245.1 Detailed Description

path pricer for performance options

`#include <ql/MonteCarlo/performanceoptionpathpricer.hpp>`

`#include <ql/Pricers/singleassetoption.hpp>`

Include dependency graph for performanceoptionpathpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

12.246 ql/MonteCarlo/performanceoptionpathpricer.hpp File Reference

12.246.1 Detailed Description

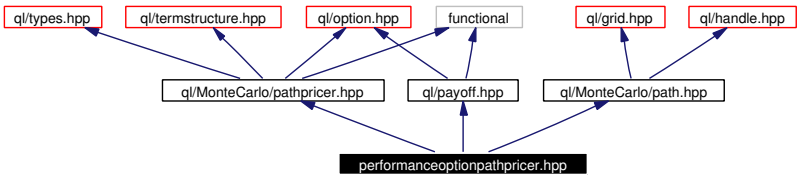
path pricer for performance option

`#include <ql/MonteCarlo/pathpricer.hpp>`

`#include <ql/MonteCarlo/path.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for performanceoptionpathpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

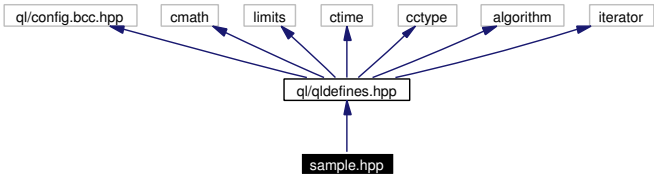
12.247 ql/MonteCarlo/sample.hpp File Reference

12.247.1 Detailed Description

weighted sample

`#include <ql/qldefines.hpp>`

Include dependency graph for sample.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MonteCarlo**

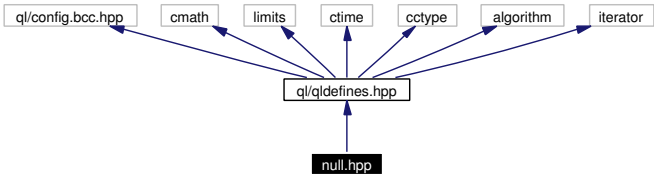
12.248 ql/null.hpp File Reference

12.248.1 Detailed Description

null values

`#include <ql/qldefines.hpp>`

Include dependency graph for null.hpp:



Namespaces

- namespace **QuantLib**

12.249 ql/numericalmethod.hpp File Reference

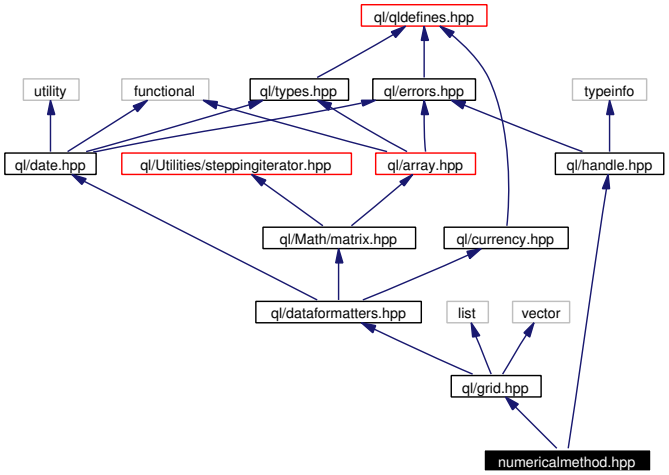
12.249.1 Detailed Description

Numerical method class.

`#include <ql/grid.hpp>`

`#include <ql/handle.hpp>`

Include dependency graph for numericalmethod.hpp:



Namespaces

- namespace **QuantLib**

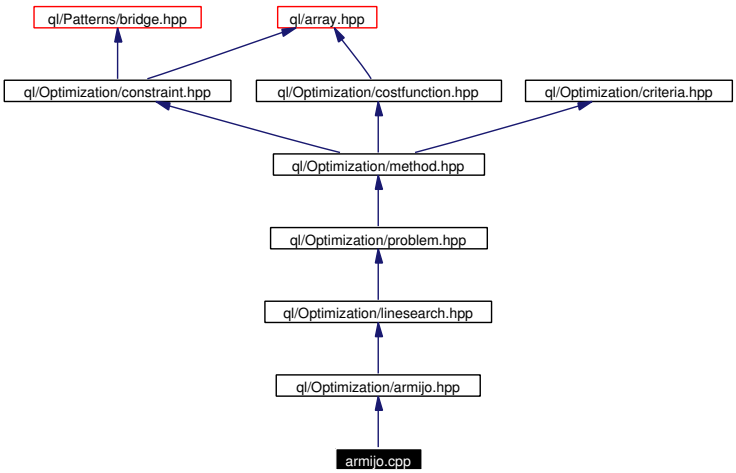
12.250 ql/Optimization/armijo.cpp File Reference

12.250.1 Detailed Description

Armijo line-search class.

`#include "ql/Optimization/armijo.hpp"`

Include dependency graph for armijo.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

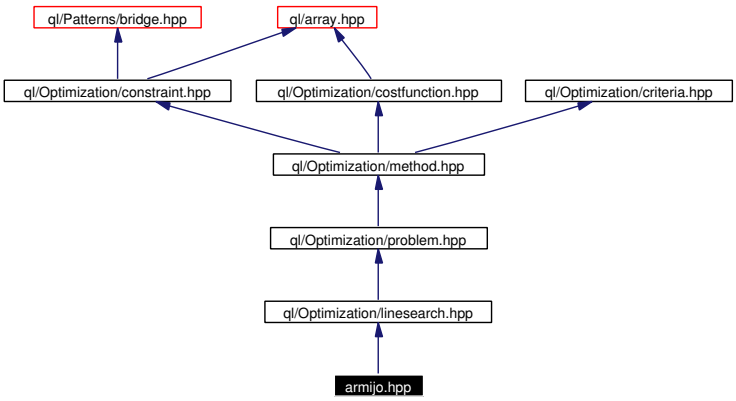
12.251 ql/Optimization/armijo.hpp File Reference

12.251.1 Detailed Description

Armijo line-search class.

`#include "ql/Optimization/linesearch.hpp"`

Include dependency graph for armijo.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

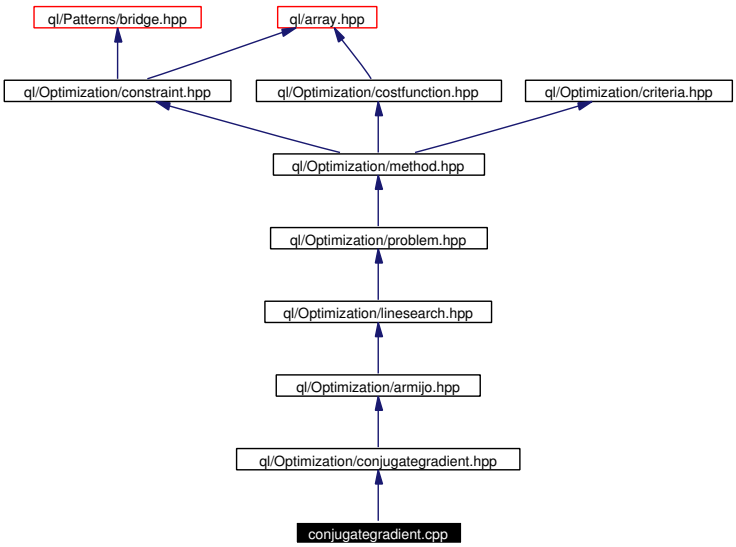
12.252 ql/Optimization/conjugategradient.cpp File Reference

12.252.1 Detailed Description

Conjugate gradient optimization method.

`#include "ql/Optimization/conjugategradient.hpp"`

Include dependency graph for conjugategradient.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

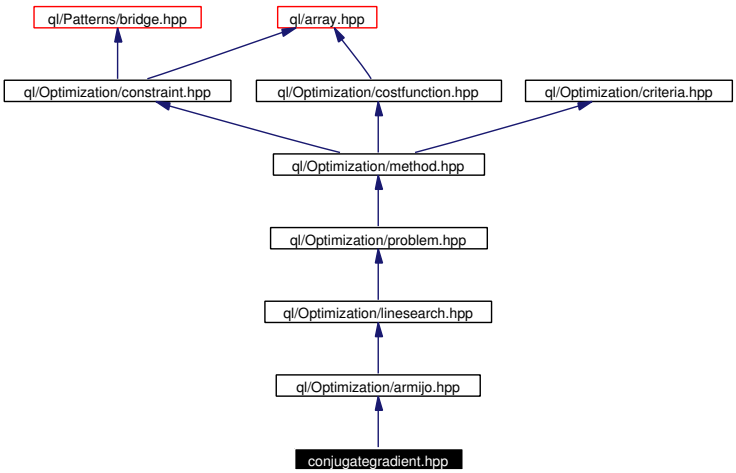
12.253 ql/Optimization/conjugategradient.hpp File Reference

12.253.1 Detailed Description

Conjugate gradient optimization method.

`#include "ql/Optimization/armijo.hpp"`

Include dependency graph for conjugategradient.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

12.254 ql/Optimization/constraint.hpp File Reference

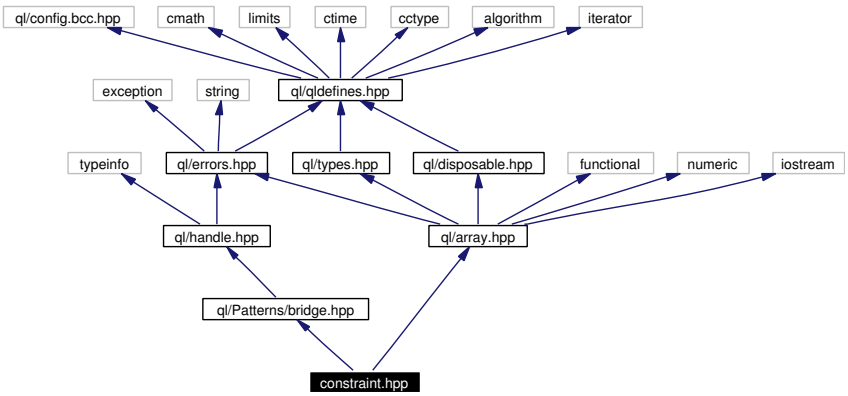
12.254.1 Detailed Description

Abstract constraint class.

`#include <ql/array.hpp>`

`#include <ql/Patterns/bridge.hpp>`

Include dependency graph for constraint.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

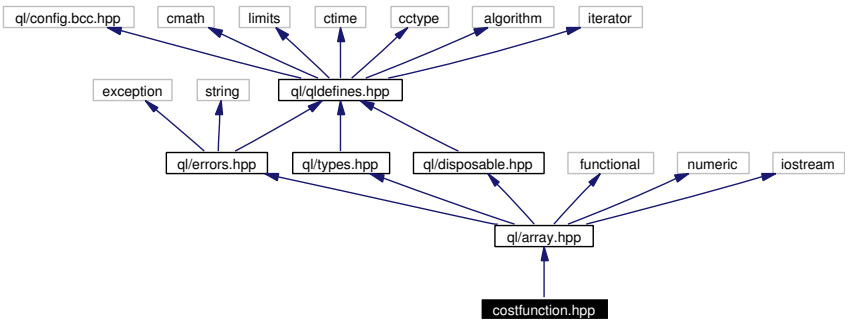
12.255 ql/Optimization/costfunction.hpp File Reference

12.255.1 Detailed Description

Optimization cost function class.

`#include <ql/array.hpp>`

Include dependency graph for costfunction.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

12.256 ql/Optimization/criteria.hpp File Reference

12.256.1 Detailed Description

Optimization criteria class.

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

12.257 ql/Optimization/leastsquare.hpp File Reference

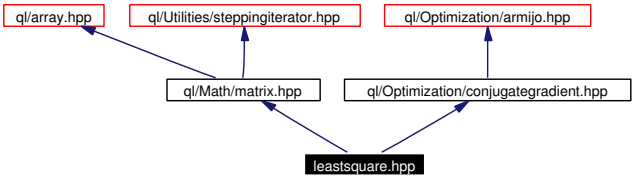
12.257.1 Detailed Description

Least square cost function.

`#include "ql/Math/matrix.hpp"`

`#include "ql/Optimization/conjugategradient.hpp"`

Include dependency graph for leastsquare.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

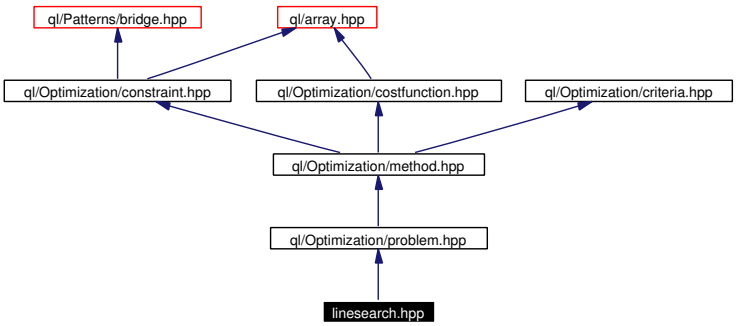
12.258 ql/Optimization/linesearch.hpp File Reference

12.258.1 Detailed Description

Line search abstract class.

`#include <ql/Optimization/problem.hpp>`

Include dependency graph for linesearch.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

12.259 ql/Optimization/method.hpp File Reference

12.259.1 Detailed Description

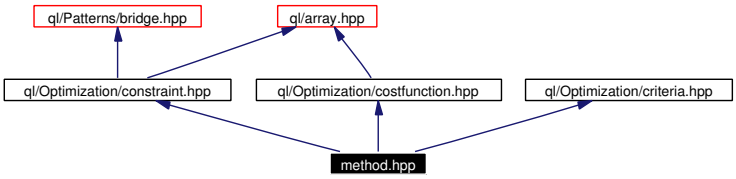
Abstract optimization method class.

`#include <ql/Optimization/constraint.hpp>`

`#include <ql/Optimization/costfunction.hpp>`

`#include <ql/Optimization/criteria.hpp>`

Include dependency graph for method.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

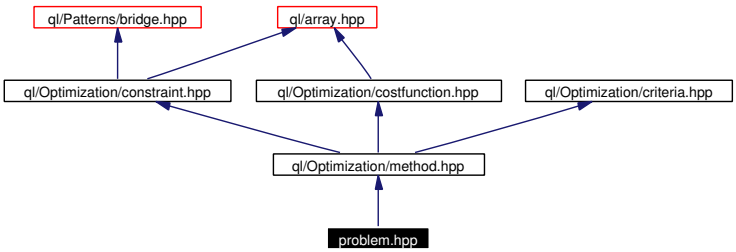
12.260 ql/Optimization/problem.hpp File Reference

12.260.1 Detailed Description

Abstract optimization class.

`#include <ql/Optimization/method.hpp>`

Include dependency graph for problem.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

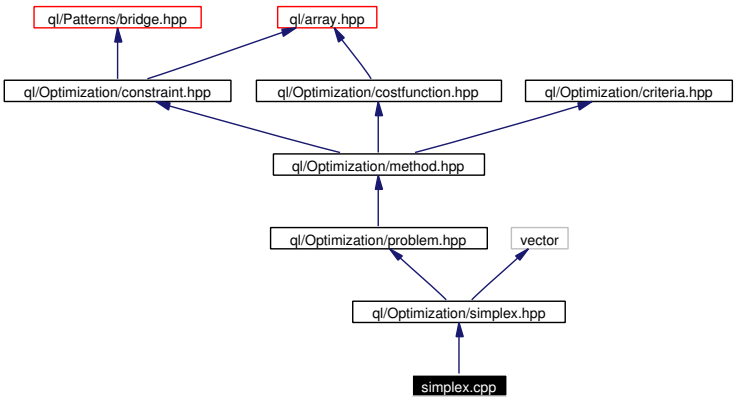
12.261 ql/Optimization/simplex.cpp File Reference

12.261.1 Detailed Description

Simplex optimization method.

`#include "ql/Optimization/simplex.hpp"`

Include dependency graph for simplex.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

12.262 ql/Optimization/simplex.hpp File Reference

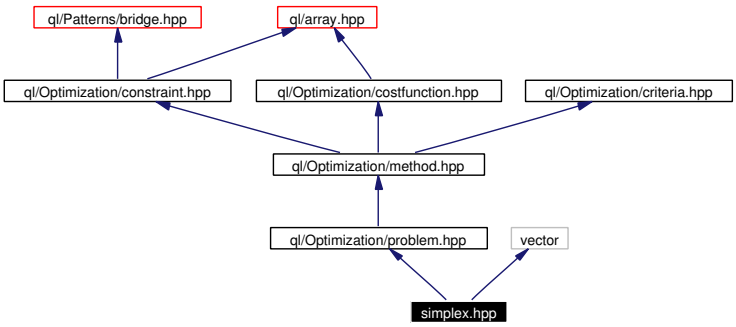
12.262.1 Detailed Description

Simplex optimization method.

`#include <ql/Optimization/problem.hpp>`

`#include <vector>`

Include dependency graph for simplex.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

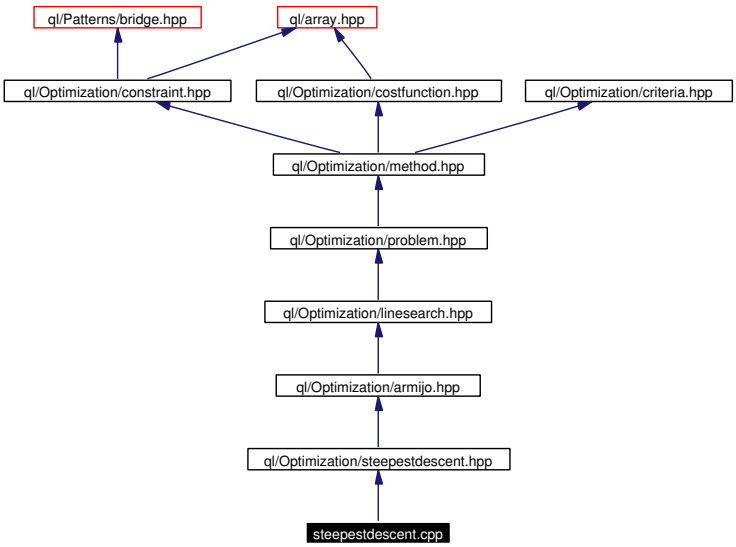
12.263 ql/Optimization/steepestdescent.cpp File Reference

12.263.1 Detailed Description

Steepest descent optimization method.

`#include "ql/Optimization/steepestdescent.hpp"`

Include dependency graph for steepestdescent.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

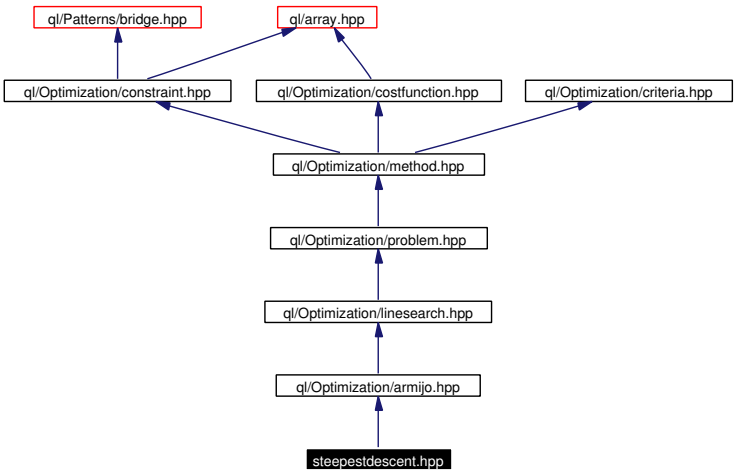
12.264 ql/Optimization/steepestdescent.hpp File Reference

12.264.1 Detailed Description

Steepest descent optimization method.

`#include <ql/Optimization/armijo.hpp>`

Include dependency graph for steepestdescent.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Optimization**

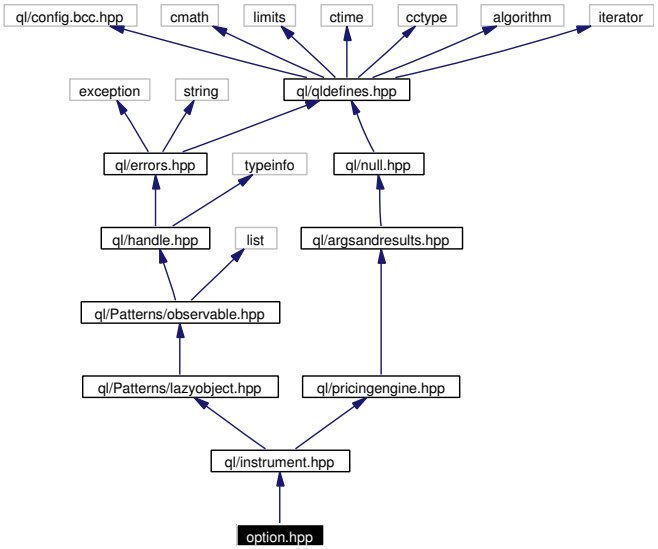
12.265 ql/option.hpp File Reference

12.265.1 Detailed Description

Base option class.

`#include <ql/instrument.hpp>`

Include dependency graph for option.hpp:



Namespaces

- namespace **QuantLib**

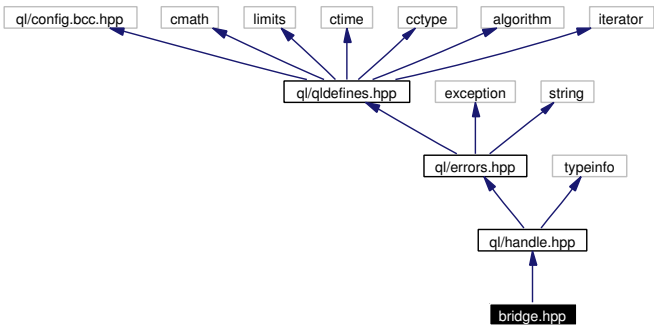
12.266 ql/Patterns/bridge.hpp File Reference

12.266.1 Detailed Description

bridge pattern (a.k.a. handle-body idiom)

`#include <ql/handle.hpp>`

Include dependency graph for bridge.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Patterns**

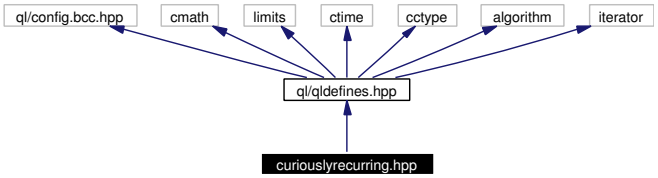
12.267 ql/Patterns/curiouslyrecurring.hpp File Reference

12.267.1 Detailed Description

Curiously recurring template pattern.

`#include <ql/qldefines.hpp>`

Include dependency graph for curiouslyrecurring.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Patterns**

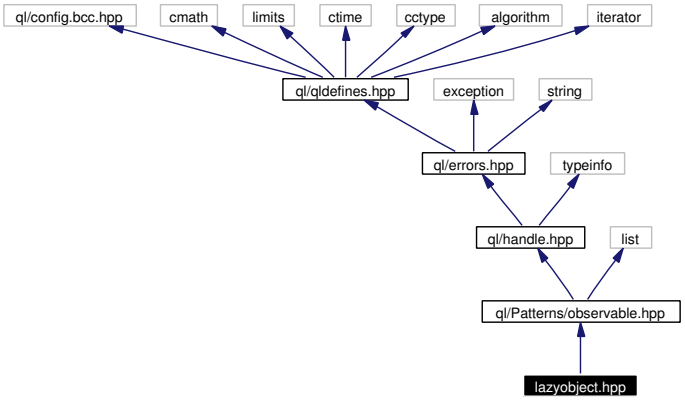
12.268 ql/Patterns/lazyobject.hpp File Reference

12.268.1 Detailed Description

framework for calculation on demand and result caching

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for lazyobject.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Patterns**

12.269 ql/Patterns/observable.hpp File Reference

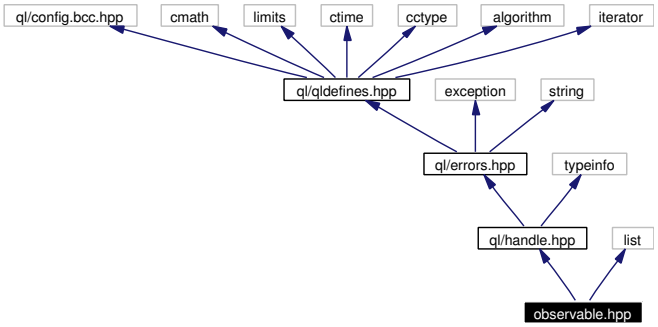
12.269.1 Detailed Description

observer/observable pattern

`#include <ql/handle.hpp>`

`#include <list>`

Include dependency graph for observable.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Patterns**

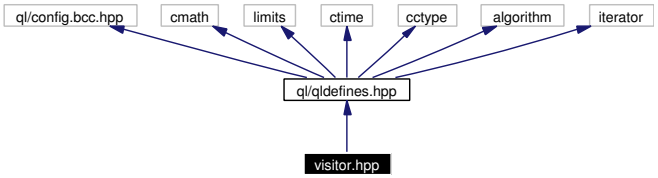
12.270 ql/Patterns/visitor.hpp File Reference

12.270.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

`#include <ql/qldefines.hpp>`

Include dependency graph for visitor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Patterns**

12.271 ql/payoff.hpp File Reference

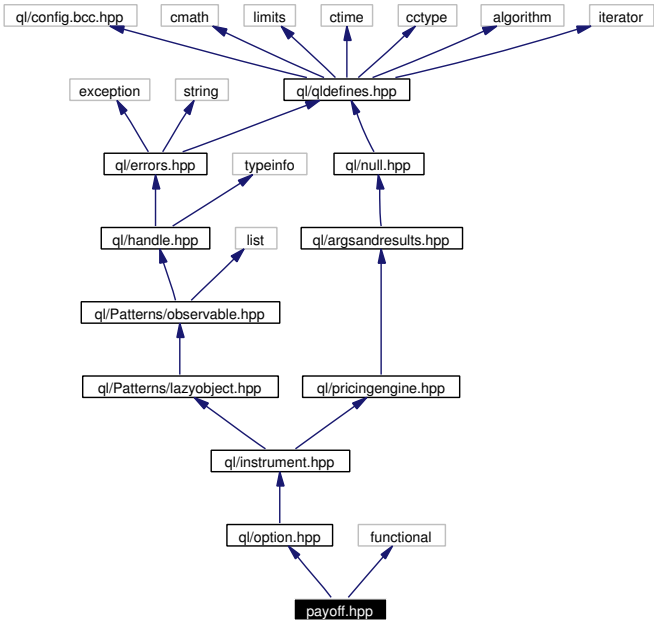
12.271.1 Detailed Description

Option payoff classes.

`#include <ql/option.hpp>`

`#include <functional>`

Include dependency graph for payoff.hpp:



Namespaces

- namespace **QuantLib**

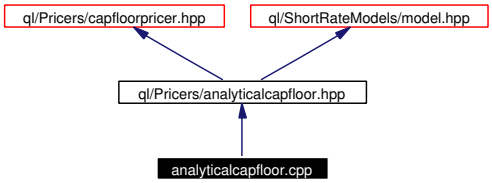
12.272 ql/Pricers/analyticalcapfloor.cpp File Reference

12.272.1 Detailed Description

Analytical pricer for caps/floors.

`#include "ql/Pricers/analyticalcapfloor.hpp"`

Include dependency graph for analyticalcapfloor.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.273 ql/Pricers/analyticalcapfloor.hpp File Reference

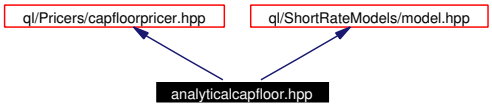
12.273.1 Detailed Description

Analytical pricer for caps/floors.

`#include <ql/Pricers/capfloorpricer.hpp>`

`#include <ql/ShortRateModels/model.hpp>`

Include dependency graph for analyticalcapfloor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

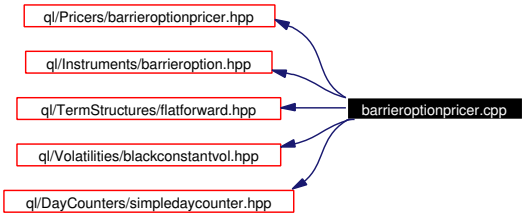
12.274 ql/Pricers/barrieroptionpricer.cpp File Reference

12.274.1 Detailed Description

barrier option pricer

```
#include <ql/Pricers/barrieroptionpricer.hpp>
#include <ql/Instruments/barrieroption.hpp>
#include <ql/TermStructures/flatforward.hpp>
#include <ql/Volatilities/blackconstantvol.hpp>
#include <ql/DayCounters/simpledaycounter.hpp>
```

Include dependency graph for barrieroptionpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.275 ql/Pricers/barrieroptionpricer.hpp File Reference

12.275.1 Detailed Description

barrier option pricer

```
#include <ql/PricingEngines/barrierengines.hpp>
#include <ql/Pricers/singleassetoption.hpp>
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for barrieroptionpricer.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.276 ql/Pricers/binaryoptionpricer.cpp File Reference

12.276.1 Detailed Description

European style cash-or-nothing option pricer.

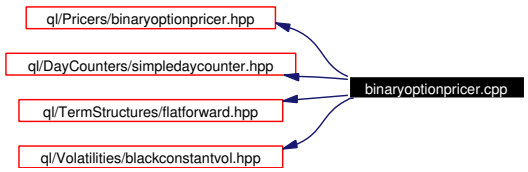
`#include <ql/Pricers/binaryoptionpricer.hpp>`

`#include <ql/DayCounters/simpledaycounter.hpp>`

`#include <ql/TermStructures/flatforward.hpp>`

`#include <ql/Volatilities/blackconstantvol.hpp>`

Include dependency graph for binaryoptionpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.277 ql/Pricers/binaryoptionpricer.hpp File Reference

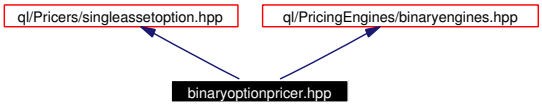
12.277.1 Detailed Description

European style cash-or-nothing option pricer.

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/PricingEngines/binaryengines.hpp>`

Include dependency graph for binaryoptionpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

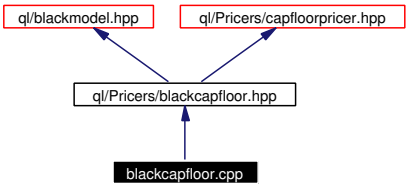
12.278 ql/Pricers/blackcapfloor.cpp File Reference

12.278.1 Detailed Description

European capfloor calculated using Black formula.

`#include "ql/Pricers/blackcapfloor.hpp"`

Include dependency graph for blackcapfloor.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.279 ql/Pricers/blackcapfloor.hpp File Reference

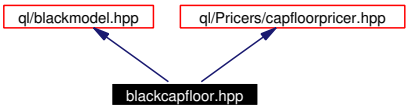
12.279.1 Detailed Description

CapFloor calculated using the Black formula.

`#include <ql/blackmodel.hpp>`

`#include <ql/Pricers/capfloorpricer.hpp>`

Include dependency graph for blackcapfloor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

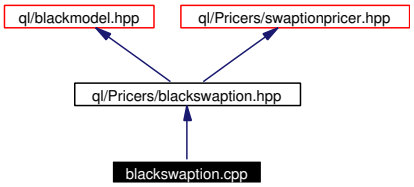
12.280 ql/Pricers/blackswaption.cpp File Reference

12.280.1 Detailed Description

European swaption calculated using Black formula.

`#include "ql/Pricers/blackswaption.hpp"`

Include dependency graph for blackswaption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.281 ql/Pricers/blackswaption.hpp File Reference

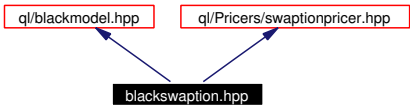
12.281.1 Detailed Description

Swaption calculated using the Black formula.

`#include <ql/blackmodel.hpp>`

`#include <ql/Pricers/swaptionpricer.hpp>`

Include dependency graph for blackswaption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

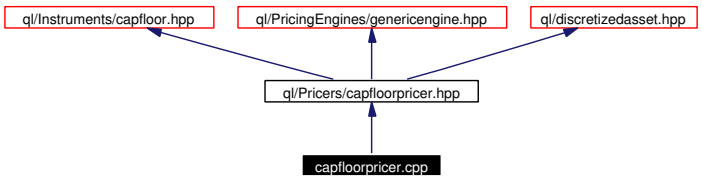
12.282 ql/Pricers/capfloorpricer.cpp File Reference

12.282.1 Detailed Description

cap and floor pricer class

`#include <ql/Pricers/capfloorpricer.hpp>`

Include dependency graph for capfloorpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

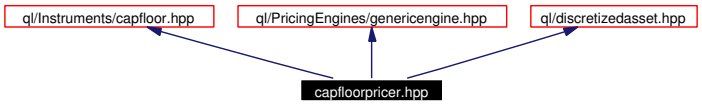
12.283 ql/Pricers/capfloorpricer.hpp File Reference

12.283.1 Detailed Description

cap and floor pricer class

```
#include <ql/Instruments/capfloor.hpp>
#include <ql/PricingEngines/genericengine.hpp>
#include <ql/discretizedasset.hpp>
```

Include dependency graph for capfloorpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.284 ql/Pricers/cliquetoption.cpp File Reference

12.284.1 Detailed Description

Cliquet option.

`#include <ql/Pricers/cliquetoption.hpp>`

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

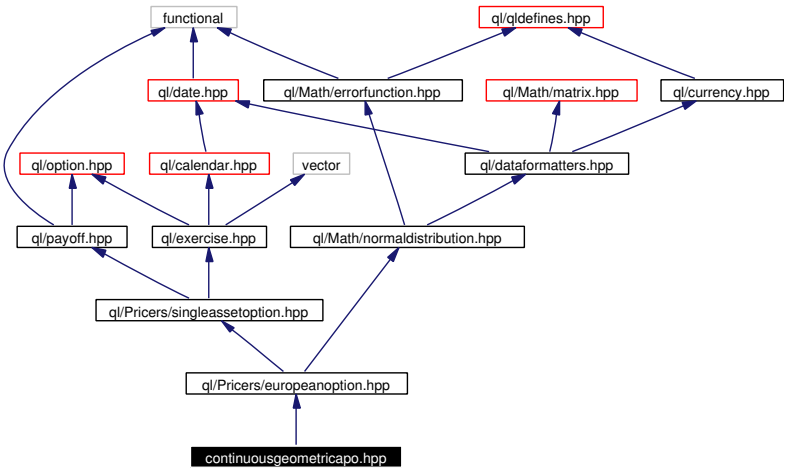
12.285 ql/Pricers/continuousgeometricapo.hpp File Reference

12.285.1 Detailed Description

Continuous Geometric Average Price Option (European exercise).

`#include <ql/Pricers/europeanoption.hpp>`

Include dependency graph for continuousgeometricapo.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.286 ql/Pricers/discretegeometricapo.cpp File Reference

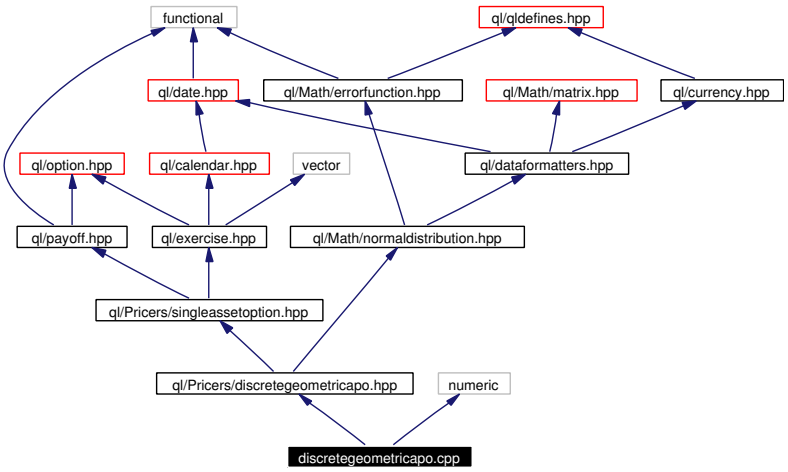
12.286.1 Detailed Description

Discrete Geometric Average Price Option.

`#include <ql/Pricers/discretegeometricapo.hpp>`

`#include <numeric>`

Include dependency graph for discretegeometricapo.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.287 ql/Pricers/discretegeometricapo.hpp File Reference

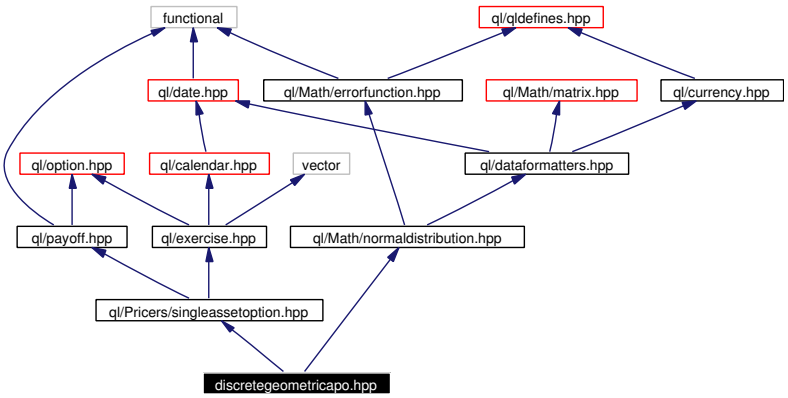
12.287.1 Detailed Description

Discrete Geometric Average Price Option.

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for discretegeometricapo.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.288 ql/Pricers/discretegeometricaso.cpp File Reference

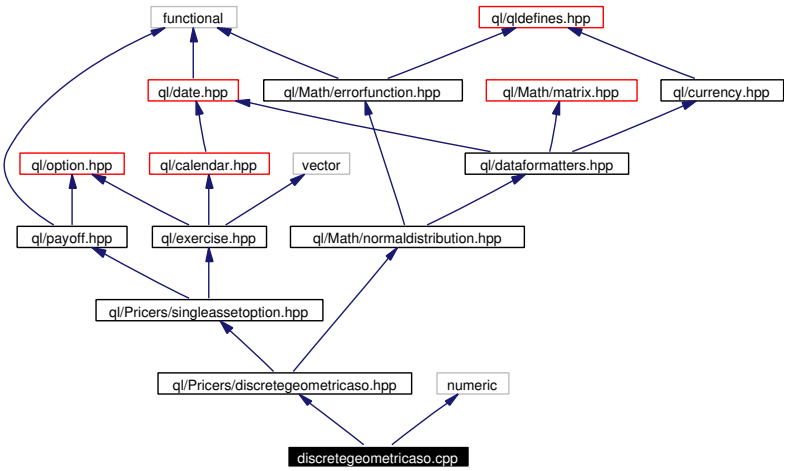
12.288.1 Detailed Description

Discrete Geometric Average Strike Option.

`#include <ql/Pricers/discretegeometricaso.hpp>`

`#include <numeric>`

Include dependency graph for discretegeometricaso.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.289 ql/Pricers/discretegeometricaso.hpp File Reference

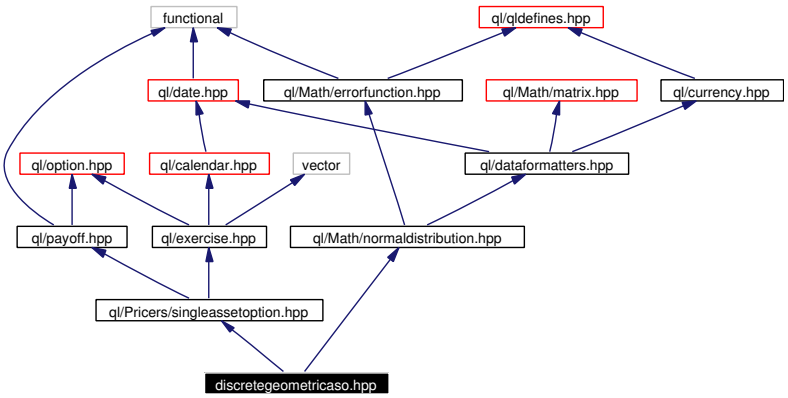
12.289.1 Detailed Description

Discrete Geometric Average Strike Option.

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for discretegeometricaso.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

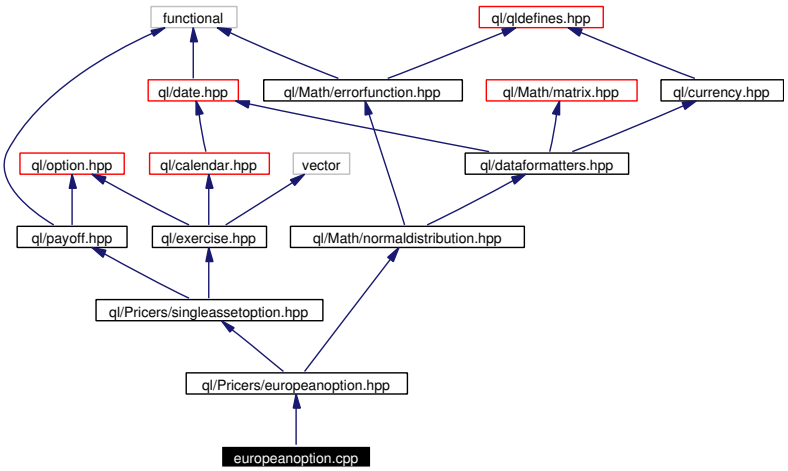
12.290 ql/Pricers/europeanoption.cpp File Reference

12.290.1 Detailed Description

european option

`#include <ql/Pricers/europeanoption.hpp>`

Include dependency graph for europeanoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.291 ql/Pricers/europeanoption.hpp File Reference

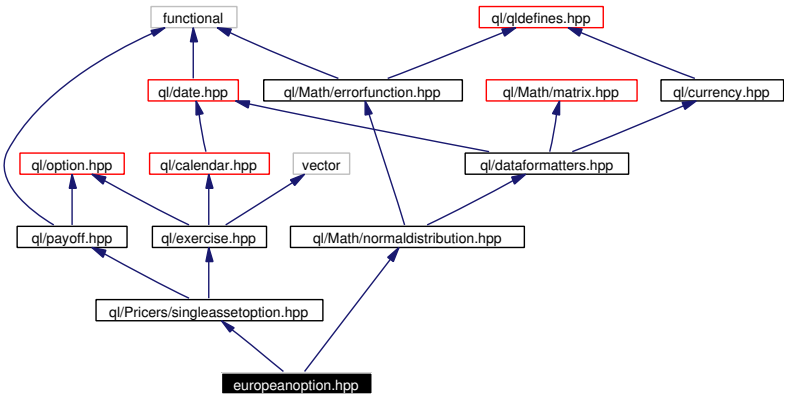
12.291.1 Detailed Description

european option

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for europeanoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.292 ql/Pricers/fdamericanoption.hpp File Reference

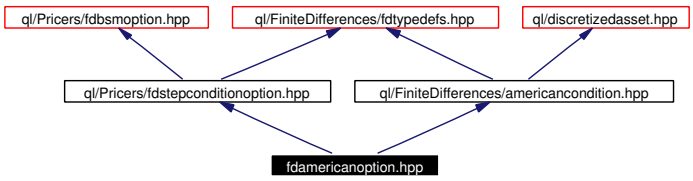
12.292.1 Detailed Description

american option

`#include <ql/Pricers/fdstepconditionoption.hpp>`

`#include <ql/FiniteDifferences/americancondition.hpp>`

Include dependency graph for fdamericanoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.293 ql/Pricers/fdbermudanoption.cpp File Reference

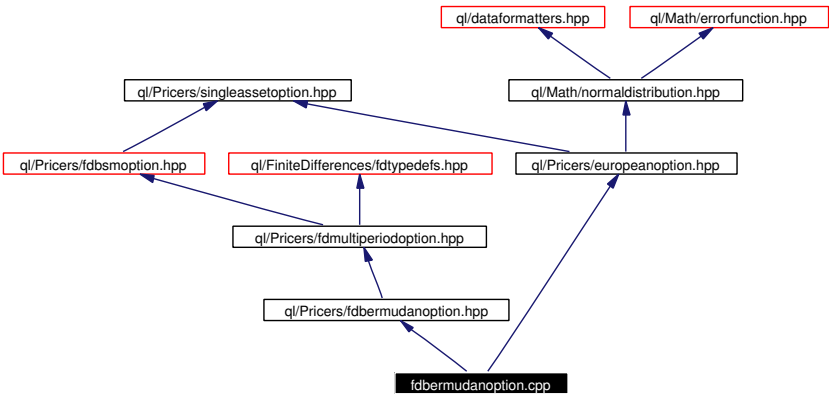
12.293.1 Detailed Description

finite-difference evaluation of Bermudan option

`#include <ql/Pricers/fdbermudanoption.hpp>`

`#include <ql/Pricers/europeanoption.hpp>`

Include dependency graph for fdbermudanoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

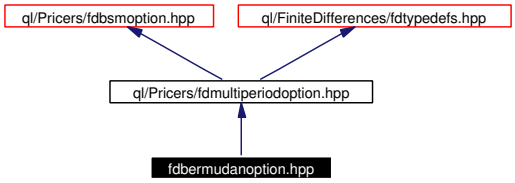
12.294 ql/Pricers/fdbermudanoption.hpp File Reference

12.294.1 Detailed Description

finite-difference evaluation of Bermudan option

`#include <ql/Pricers/fdmultiperiodoption.hpp>`

Include dependency graph for fdbermudanoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.295 ql/Pricers/fdbsmoption.cpp File Reference

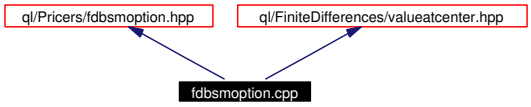
12.295.1 Detailed Description

common code for numerical option evaluation

`#include <ql/Pricers/fdbsmoption.hpp>`

`#include <ql/FiniteDifferences/valueatcenter.hpp>`

Include dependency graph for fdbsmoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.296 ql/Pricers/fdbsmoption.hpp File Reference

12.296.1 Detailed Description

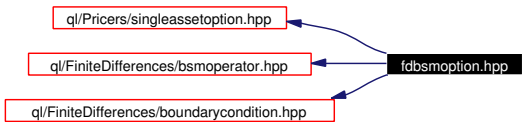
common code for numerical option evaluation

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/FiniteDifferences/bsmoperator.hpp>`

`#include <ql/FiniteDifferences/boundarycondition.hpp>`

Include dependency graph for fdbsmoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

Defines

- #define **QL_NUM_OPT_MIN_GRID_POINTS** 10

  This is a safety check to be sure we have enough grid points.

- #define **QL_NUM_OPT_GRID_POINTS_PER_YEAR** 2

  This is a safety check to be sure we have enough grid points.

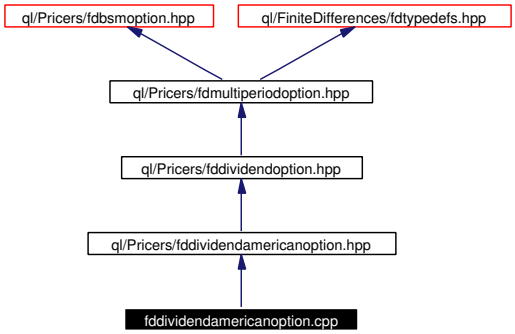
12.297 ql/Pricers/fddividendamericanoption.cpp File Reference

12.297.1 Detailed Description

american option with discrete deterministic dividends

`#include <ql/Pricers/fddividendamericanoption.hpp>`

Include dependency graph for fddividendamericanoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

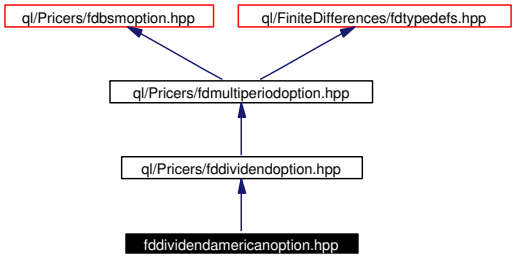
12.298 ql/Pricers/fddividendamericanoption.hpp File Reference

12.298.1 Detailed Description

american option with discrete deterministic dividends

`#include <ql/Pricers/fddividendoption.hpp>`

Include dependency graph for fddividendamericanoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

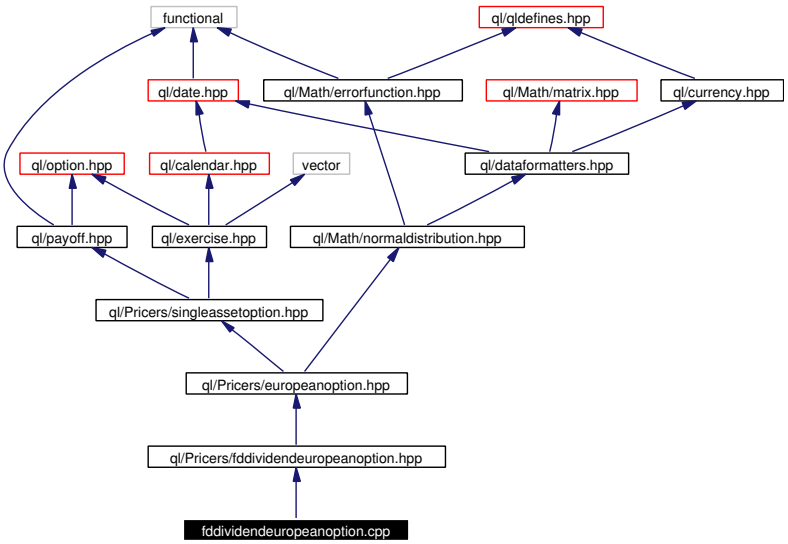
12.299 ql/Pricers/fddividendeuropeanoption.cpp File Reference

12.299.1 Detailed Description

european option with discrete deterministic dividends

`#include <ql/Pricers/fddividendeuropeanoption.hpp>`

Include dependency graph for fddividendeuropeanoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

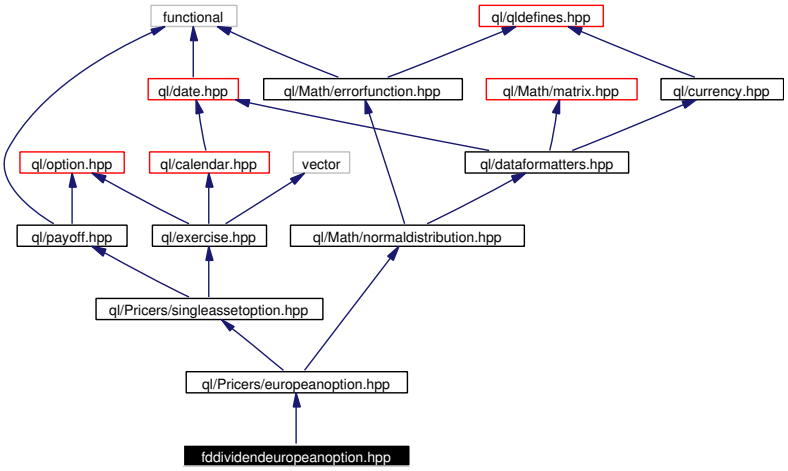
12.300 ql/Pricers/fddividendeuropeanoption.hpp File Reference

12.300.1 Detailed Description

european option with discrete deterministic dividends

`#include <ql/Pricers/europeanoption.hpp>`

Include dependency graph for fddividendeuropeanoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

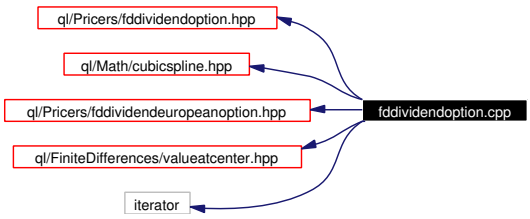
12.301 ql/Pricers/fddividendoption.cpp File Reference

12.301.1 Detailed Description

base class for options with dividends

```
#include <ql/Pricers/fddividendoption.hpp>
#include <ql/Math/cubicspline.hpp>
#include <ql/Pricers/fddividendeuropeanoption.hpp>
#include <ql/FiniteDifferences/valueatcenter.hpp>
#include <iterator>
```

Include dependency graph for fddividendoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

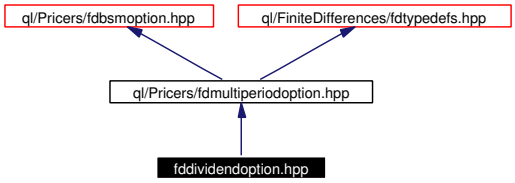
12.302 ql/Pricers/fddividendoption.hpp File Reference

12.302.1 Detailed Description

base class for option with dividends

`#include <ql/Pricers/fdmultiperiodoption.hpp>`

Include dependency graph for fddividendoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

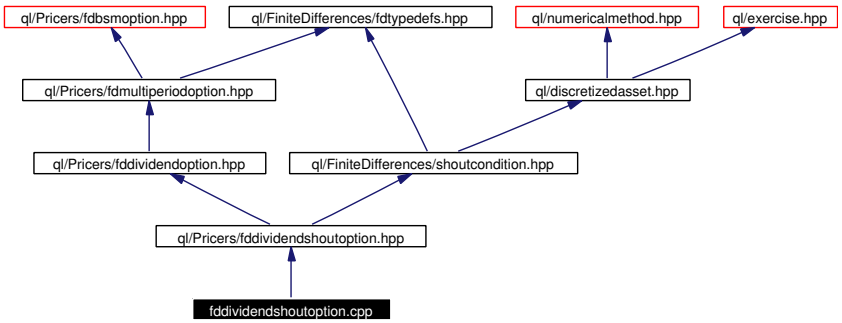
12.303 ql/Pricers/fddividendshoutoption.cpp File Reference

12.303.1 Detailed Description

base class for shout options with dividends

`#include <ql/Pricers/fddividendshoutoption.hpp>`

Include dependency graph for fddividendshoutoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.304 ql/Pricers/fddividendshoutoption.hpp File Reference

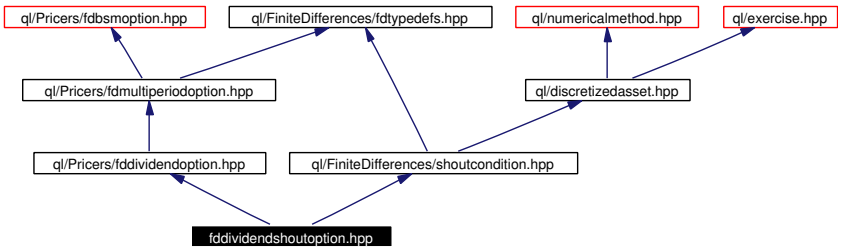
12.304.1 Detailed Description

base class for shout option with dividends

`#include <ql/Pricers/fddividendoption.hpp>`

`#include <ql/FiniteDifferences/shoutcondition.hpp>`

Include dependency graph for fddividendshoutoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.305 ql/Pricers/fdeuropean.cpp File Reference

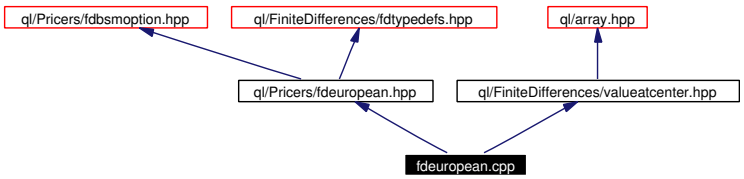
12.305.1 Detailed Description

Example of European option calculated using finite differences.

`#include <ql/Pricers/fdeuropean.hpp>`

`#include <ql/FiniteDifferences/valueatcenter.hpp>`

Include dependency graph for fdeuropean.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.306 ql/Pricers/fdeuropean.hpp File Reference

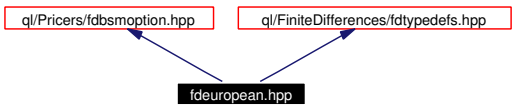
12.306.1 Detailed Description

Example of European option calculated using finite differences.

`#include <ql/Pricers/fdbsmoption.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

Include dependency graph for fdeuropean.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.307 ql/Pricers/fdmultiperiodoption.cpp File Reference

12.307.1 Detailed Description

base class for option with events happening at different periods

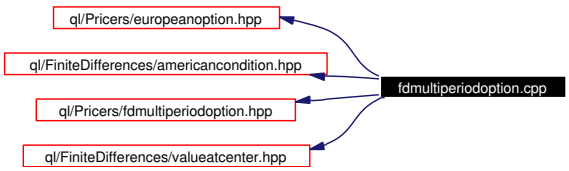
`#include <ql/Pricers/europeanoption.hpp>`

`#include <ql/FiniteDifferences/americancondition.hpp>`

`#include <ql/Pricers/fdmultiperiodoption.hpp>`

`#include <ql/FiniteDifferences/valueatcenter.hpp>`

Include dependency graph for fdmultiperiodoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.308 ql/Pricers/fdmultiperiodoption.hpp File Reference

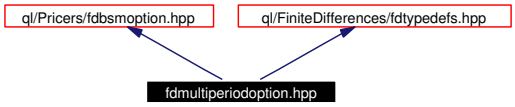
12.308.1 Detailed Description

base class for option with events happening at different periods

`#include <ql/Pricers/fdbsmoption.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

Include dependency graph for fdmultiperiodoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.309 ql/Pricers/fdshoutoption.hpp File Reference

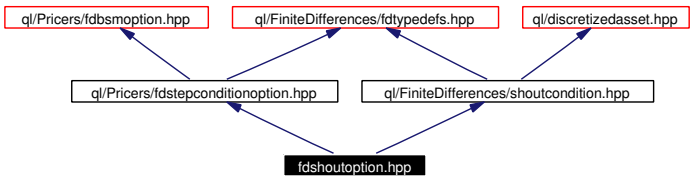
12.309.1 Detailed Description

shout option

`#include <ql/Pricers/fdstepconditionoption.hpp>`

`#include <ql/FiniteDifferences/shoutcondition.hpp>`

Include dependency graph for fdshoutoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.310 ql/Pricers/fdstepconditionoption.cpp File Reference

12.310.1 Detailed Description

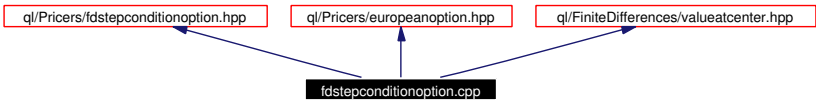
Option requiring additional code to be executed at each time step.

`#include <ql/Pricers/fdstepconditionoption.hpp>`

`#include <ql/Pricers/europeanoption.hpp>`

`#include <ql/FiniteDifferences/valueatcenter.hpp>`

Include dependency graph for fdstepconditionoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.311 ql/Pricers/fdstepconditionoption.hpp File Reference

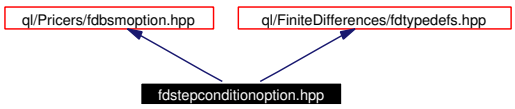
12.311.1 Detailed Description

Option requiring additional code to be executed at each time step.

`#include <ql/Pricers/fdbsmoption.hpp>`

`#include <ql/FiniteDifferences/fdtypedefs.hpp>`

Include dependency graph for fdstepconditionoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.312 ql/Pricers/jamshidianswaption.cpp File Reference

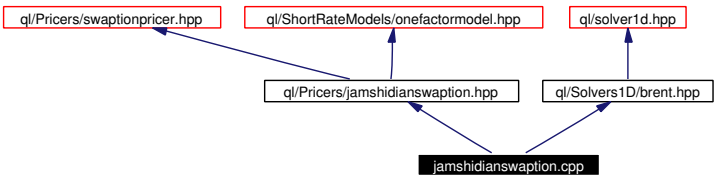
12.312.1 Detailed Description

Swaption pricer using Jamshidian's decomposition.

`#include "ql/Pricers/jamshidianswaption.hpp"`

`#include "ql/Solvers1D/brent.hpp"`

Include dependency graph for jamshidianswaption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.313 ql/Pricers/jamshidianswaption.hpp File Reference

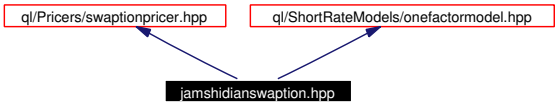
12.313.1 Detailed Description

Swaption pricer using Jamshidian's decomposition.

`#include <ql/Pricers/swaptionpricer.hpp>`

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for jamshidianswaption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.314 ql/Pricers/mcbasket.cpp File Reference

12.314.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

```
#include <ql/handle.hpp>
#include <ql/MonteCarlo/basketpathpricer.hpp>
#include <ql/Pricers/mcbasket.hpp>
```

Include dependency graph for mcbasket.cpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.315 ql/Pricers/mcbasket.hpp File Reference

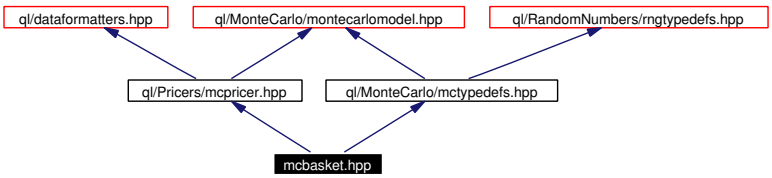
12.315.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/MonteCarlo/mctypedefs.hpp>`

Include dependency graph for mcbasket.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.316 ql/Pricers/mccliquetoption.cpp File Reference

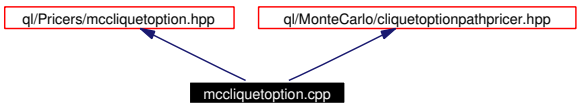
12.316.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

`#include <ql/Pricers/mccliquetoption.hpp>`

`#include <ql/MonteCarlo/cliquetoptionpathpricer.hpp>`

Include dependency graph for mccliquetoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.317 ql/Pricers/mccliquetoption.hpp File Reference

12.317.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mccliquetoption.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

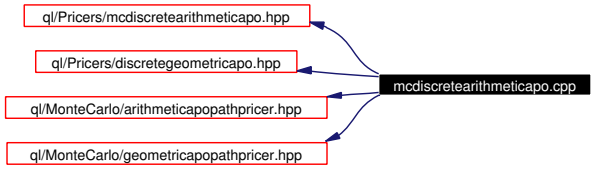
12.318 ql/Pricers/mcdiscretearithmeticapo.cpp File Reference

12.318.1 Detailed Description

Discrete Arithmetic Average Price Option.

```
#include <ql/Pricers/mcdiscretearithmeticapo.hpp>
#include <ql/Pricers/discretegeometricapo.hpp>
#include <ql/MonteCarlo/arithmeticapopathpricer.hpp>
#include <ql/MonteCarlo/geometricapopathpricer.hpp>
```

Include dependency graph for mcdiscretearithmeticapo.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.319 ql/Pricers/mcdiscretearithmeticapo.hpp File Reference

12.319.1 Detailed Description

Discrete Arithmetic Average Price Option.

`#include <ql/option.hpp>`

`#include <ql/types.hpp>`

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/MonteCarlo/mctypedefs.hpp>`

Include dependency graph for mcdiscretearithmeticapo.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

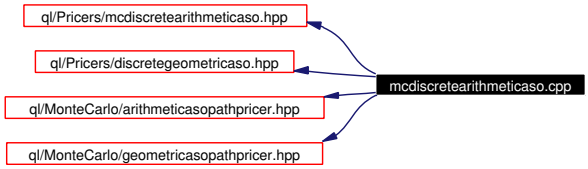
12.320 ql/Pricers/mcdiscretearithmeticaso.cpp File Reference

12.320.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/Pricers/mcdiscretearithmeticaso.hpp>
#include <ql/Pricers/discretegeometricaso.hpp>
#include <ql/MonteCarlo/arithmeticasopathpricer.hpp>
#include <ql/MonteCarlo/geometricasopathpricer.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.321 ql/Pricers/mcdiscretearithmeticaso.hpp File Reference

12.321.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.322 ql/Pricers/mceuropean.cpp File Reference

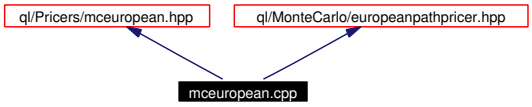
12.322.1 Detailed Description

simple example of Monte Carlo pricer

`#include <ql/Pricers/mceuropean.hpp>`

`#include <ql/MonteCarlo/europeanpathpricer.hpp>`

Include dependency graph for mceuropean.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

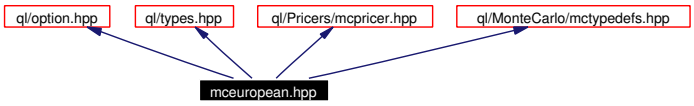
12.323 ql/Pricers/mceuropean.hpp File Reference

12.323.1 Detailed Description

simple example of Monte Carlo pricer

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mceuropean.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.324 ql/Pricers/mceverest.cpp File Reference

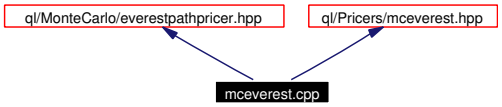
12.324.1 Detailed Description

Everest-type option pricer

`#include <ql/MonteCarlo/everestpathpricer.hpp>`

`#include <ql/Pricers/mceverest.hpp>`

Include dependency graph for mceverest.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.325 ql/Pricers/mceverest.hpp File Reference

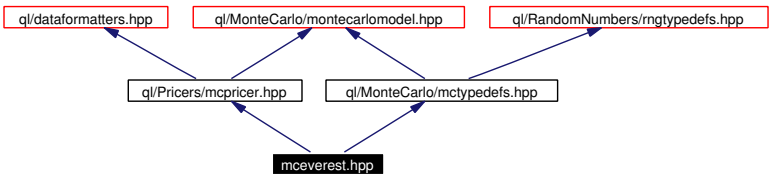
12.325.1 Detailed Description

Everest-type option pricer

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/MonteCarlo/mctypedefs.hpp>`

Include dependency graph for mceverest.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

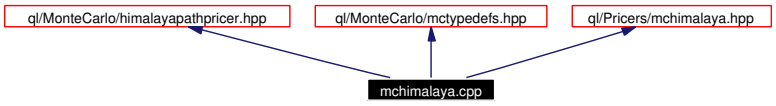
12.326 ql/Pricers/mchimalaya.cpp File Reference

12.326.1 Detailed Description

Himalayan-type option pricer.

```
#include <ql/MonteCarlo/himalayapathpricer.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
#include <ql/Pricers/mchimalaya.hpp>
```

Include dependency graph for mchimalaya.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

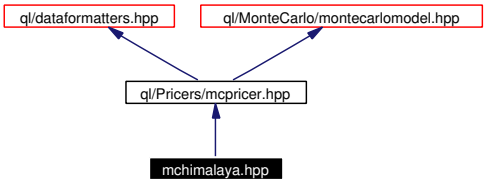
12.327 ql/Pricers/mchimalaya.hpp File Reference

12.327.1 Detailed Description

Himalayan-type option pricer.

`#include <ql/Pricers/mcpricer.hpp>`

Include dependency graph for mchimalaya.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.328 ql/Pricers/mcmaxbasket.cpp File Reference

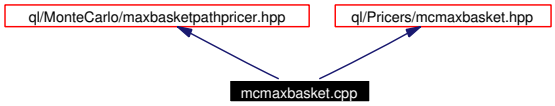
12.328.1 Detailed Description

Max Basket Monte Carlo pricer.

`#include <ql/MonteCarlo/maxbasketpathpricer.hpp>`

`#include <ql/Pricers/mcmaxbasket.hpp>`

Include dependency graph for mcmaxbasket.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.329 ql/Pricers/mcmaxbasket.hpp File Reference

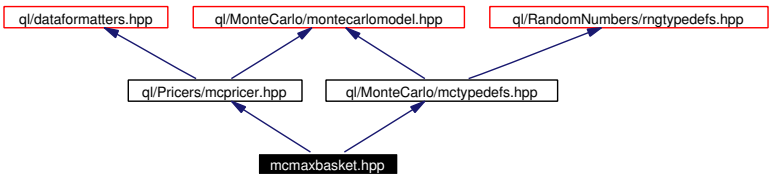
12.329.1 Detailed Description

Max Basket Monte Carlo pricer.

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/MonteCarlo/mctypedefs.hpp>`

Include dependency graph for mcmaxbasket.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

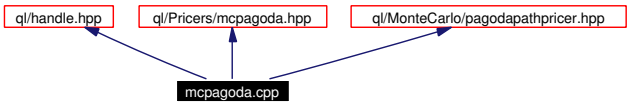
12.330 ql/Pricers/mcpagoda.cpp File Reference

12.330.1 Detailed Description

Roofed multi asset Asian option.

```
#include <ql/handle.hpp>
#include <ql/Pricers/mcpagoda.hpp>
#include <ql/MonteCarlo/pagodapathpricer.hpp>
```

Include dependency graph for mcpagoda.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.331 ql/Pricers/mcpagoda.hpp File Reference

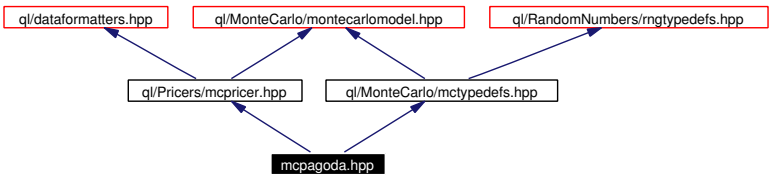
12.331.1 Detailed Description

Roofed multi asset Asian option.

`#include <ql/Pricers/mcpricer.hpp>`

`#include <ql/MonteCarlo/mctypedefs.hpp>`

Include dependency graph for mcpagoda.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.332 ql/Pricers/mcperformanceoption.cpp File Reference

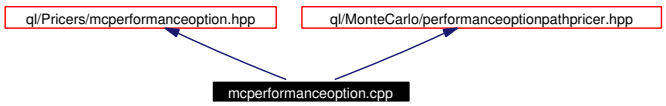
12.332.1 Detailed Description

Performance option priced with Monte Carlo simulation.

`#include <ql/Pricers/mcperformanceoption.hpp>`

`#include <ql/MonteCarlo/performanceoptionpathpricer.hpp>`

Include dependency graph for mcperformanceoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.333 ql/Pricers/mcperformanceoption.hpp File Reference

12.333.1 Detailed Description

Performance option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
#include <ql/types.hpp>
#include <ql/Pricers/mcpricer.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcperformanceoption.hpp:

Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.334 ql/Pricers/mcpricer.hpp File Reference

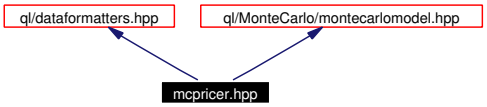
12.334.1 Detailed Description

base class for Monte Carlo pricers

`#include <ql/dataformatters.hpp>`

`#include <ql/MonteCarlo/montecarlomodel.hpp>`

Include dependency graph for mcpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

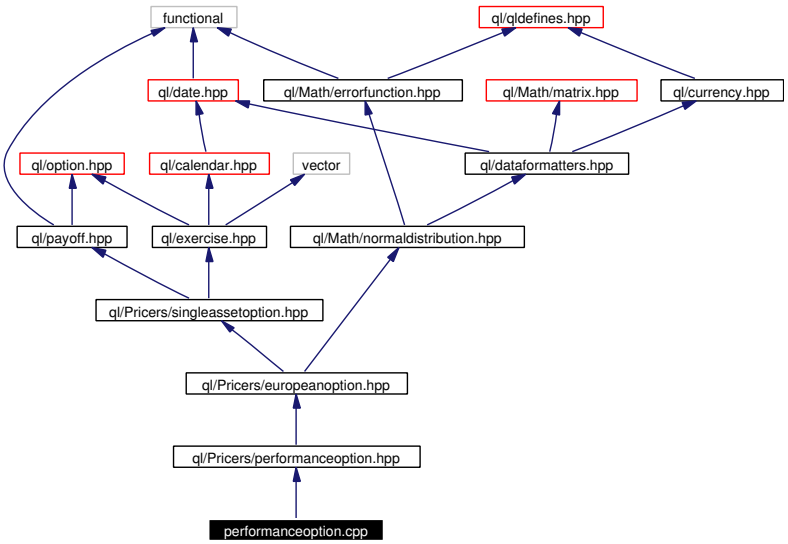
12.335 ql/Pricers/performanceoption.cpp File Reference

12.335.1 Detailed Description

Performance option.computed using Monte Carlo simulation.

`#include <ql/Pricers/performanceoption.hpp>`

Include dependency graph for performanceoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

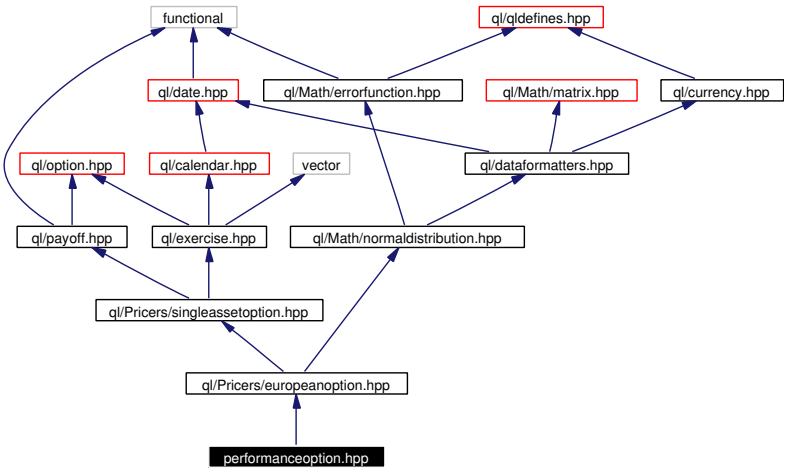
12.336 ql/Pricers/performanceoption.hpp File Reference

12.336.1 Detailed Description

Performance option.

`#include <ql/Pricers/europeanoption.hpp>`

Include dependency graph for performanceoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.337 ql/Pricers/singleassetoption.cpp File Reference

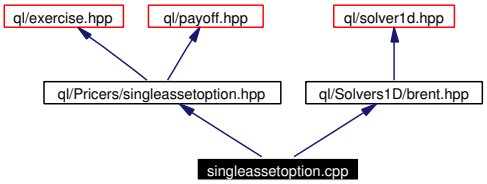
12.337.1 Detailed Description

common code for option evaluation

`#include <ql/Pricers/singleassetoption.hpp>`

`#include <ql/Solvers1D/brent.hpp>`

Include dependency graph for singleassetoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.338 ql/Pricers/singleassetoption.hpp File Reference

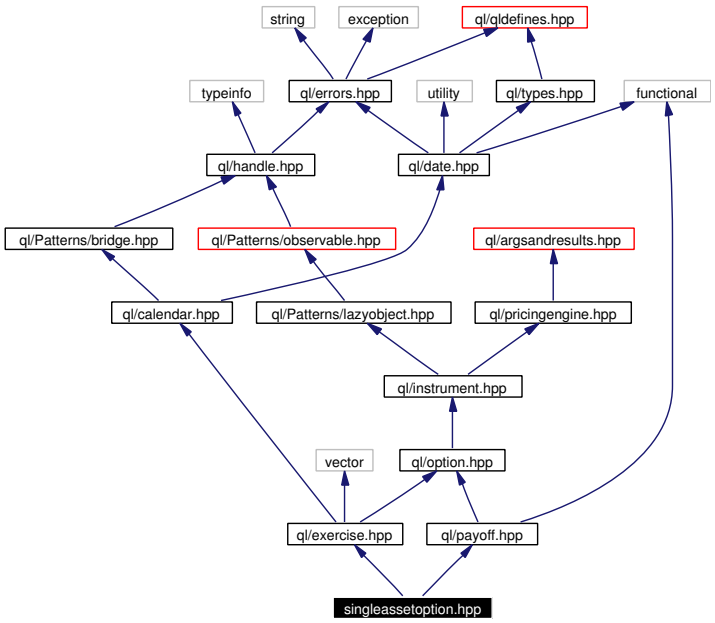
12.338.1 Detailed Description

common code for option evaluation

`#include <ql/exercise.hpp>`

`#include <ql/payoff.hpp>`

Include dependency graph for singleassetoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

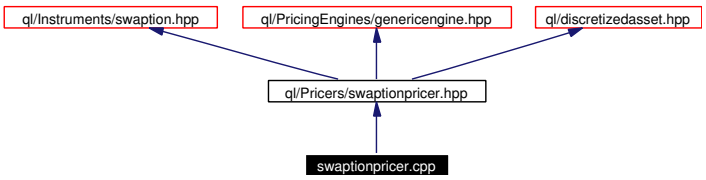
12.339 ql/Pricers/swaptionpricer.cpp File Reference

12.339.1 Detailed Description

Swaption pricer.

`#include <ql/Pricers/swaptionpricer.hpp>`

Include dependency graph for swaptionpricer.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.340 ql/Pricers/swaptionpricer.hpp File Reference

12.340.1 Detailed Description

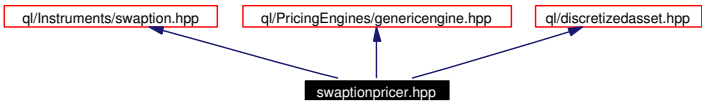
Swaption pricer class.

`#include <ql/Instruments/swaption.hpp>`

`#include <ql/PricingEngines/genericengine.hpp>`

`#include <ql/discretizedasset.hpp>`

Include dependency graph for swaptionpricer.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

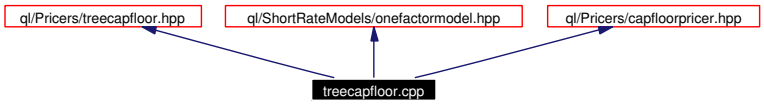
12.341 ql/Pricers/treecapfloor.cpp File Reference

12.341.1 Detailed Description

Cap/Floor calculated using a tree.

```
#include "ql/Pricers/treecapfloor.hpp"
#include "ql/ShortRateModels/onefactormodel.hpp"
#include <ql/Pricers/capfloorpricer.hpp>
```

Include dependency graph for treecapfloor.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.342 ql/Pricers/treecapfloor.hpp File Reference

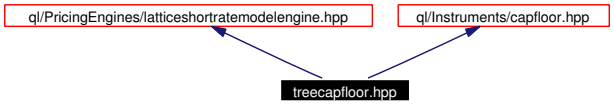
12.342.1 Detailed Description

Cap/Floor calculated using a tree.

`#include <ql/PricingEngines/latticeshortratemodelengine.hpp>`

`#include <ql/Instruments/capfloor.hpp>`

Include dependency graph for treecapfloor.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.343 ql/Pricers/treeswaption.cpp File Reference

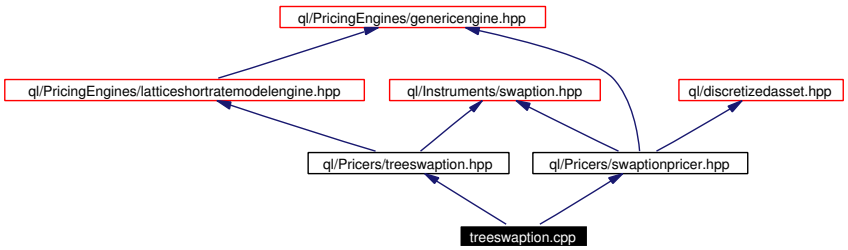
12.343.1 Detailed Description

European swaption computed on a lattice.

`#include "ql/Pricers/treeswaption.hpp"`

`#include <ql/Pricers/swaptionpricer.hpp>`

Include dependency graph for treeswaption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

12.344 ql/Pricers/treeswaption.hpp File Reference

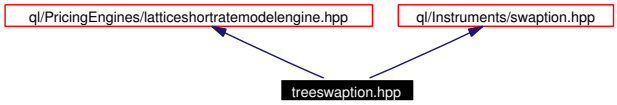
12.344.1 Detailed Description

Swaption computed using a lattice.

`#include <ql/PricingEngines/latticeshortratemodelengine.hpp>`

`#include <ql/Instruments/swaption.hpp>`

Include dependency graph for treeswaption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Pricers**

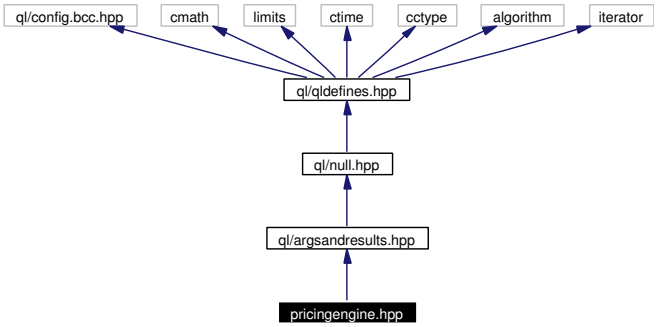
12.345 ql/pricingengine.hpp File Reference

12.345.1 Detailed Description

Base class for pricing engines.

`#include <ql/argsandresults.hpp>`

Include dependency graph for pricingengine.hpp:



Namespaces

- namespace **QuantLib**

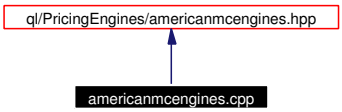
12.346 ql/PricingEngines/americanmcengines.cpp File Reference

12.346.1 Detailed Description

Least-square Monte Carlo engines.

`#include <ql/PricingEngines/americanmcengines.hpp>`

Include dependency graph for americanmcengines.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

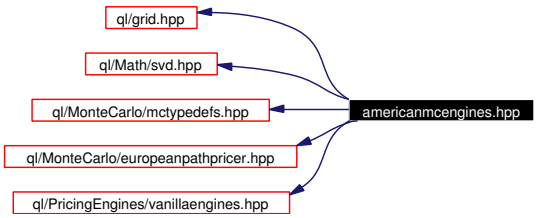
12.347 ql/PricingEngines/americanmcengines.hpp File Reference

12.347.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/grid.hpp>
#include <ql/Math/svd.hpp>
#include <ql/MonteCarlo/mctypedefs.hpp>
#include <ql/MonteCarlo/europeanpathpricer.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
```

Include dependency graph for americanmcengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

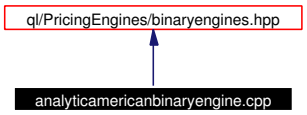
12.348 ql/PricingEngines/analyticamericanbinaryengine.cpp File Reference

12.348.1 Detailed Description

American binary option engine using analytic formulas.

`#include <ql/PricingEngines/binaryengines.hpp>`

Include dependency graph for analyticamericanbinaryengine.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

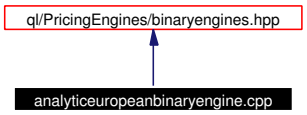
12.349 ql/PricingEngines/analyticeuropeanbinaryengine.cpp File Reference

12.349.1 Detailed Description

European binary option engine.

`#include <ql/PricingEngines/binaryengines.hpp>`

Include dependency graph for analyticeuropeanbinaryengine.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

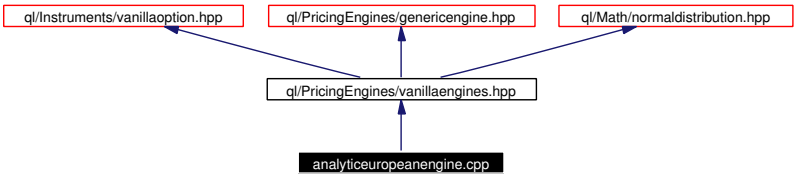
12.350 ql/PricingEngines/analyticeuropeanengine.cpp File Reference

12.350.1 Detailed Description

European option engine using analytic formulas.

`#include <ql/PricingEngines/vanillaengines.hpp>`

Include dependency graph for analyticeuropeanengine.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

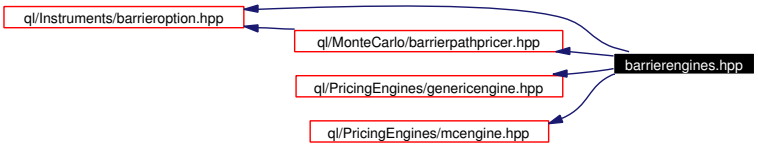
12.351 ql/PricingEngines/barrierengines.hpp File Reference

12.351.1 Detailed Description

Barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
#include <ql/PricingEngines/genericengine.hpp>
#include <ql/PricingEngines/mcengine.hpp>
#include <ql/MonteCarlo/barrierpathpricer.hpp>
#include <ql/MonteCarlo/biasedbarrierpathpricer.hpp>
```

Include dependency graph for barrierengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.352 ql/PricingEngines/binaryengines.hpp File Reference

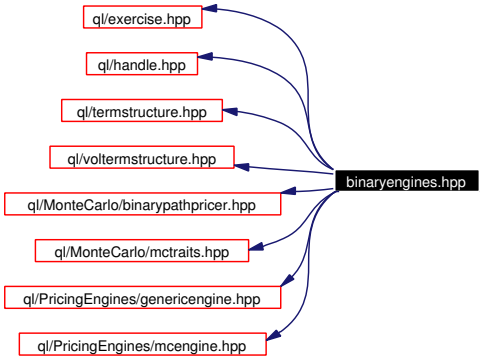
12.352.1 Detailed Description

binary option engines

Based on the Vanilla Engine pattern

```
#include <ql/exercise.hpp>
#include <ql/handle.hpp>
#include <ql/termstructure.hpp>
#include <ql/voltermstructure.hpp>
#include <ql/MonteCarlo/binarypathpricer.hpp>
#include <ql/MonteCarlo/mctraits.hpp>
#include <ql/PricingEngines/genericengine.hpp>
#include <ql/PricingEngines/mcengine.hpp>
```

Include dependency graph for binaryengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

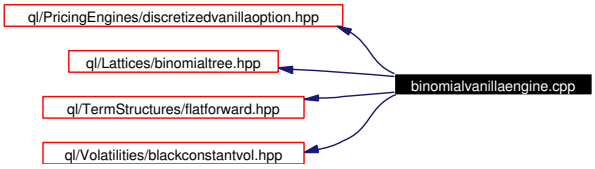
12.353 ql/PricingEngines/binomialvanillaengine.cpp File Reference

12.353.1 Detailed Description

Vanilla option engine using binomial tree.

```
#include <ql/PricingEngines/discretizedvanillaoption.hpp>
#include <ql/Lattices/binomialtree.hpp>
#include <ql/TermStructures/flatforward.hpp>
#include <ql/Volatilities/blackconstantvol.hpp>
```

Include dependency graph for binomialvanillaengine.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.354 ql/PricingEngines/cliquetengines.hpp File Reference

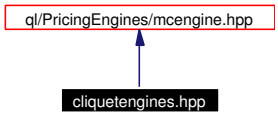
12.354.1 Detailed Description

Cliquet option engines.

`#include <ql/Instruments/cliquetoption.hpp>`

`#include <ql/PricingEngines/mcengine.hpp>`

Include dependency graph for cliquetengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.355 ql/PricingEngines/discretizedvanillaoption.cpp File Reference

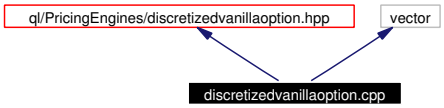
12.355.1 Detailed Description

discretized vanilla option

`#include <ql/PricingEngines/discretizedvanillaoption.hpp>`

`#include <vector>`

Include dependency graph for discretizedvanillaoption.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

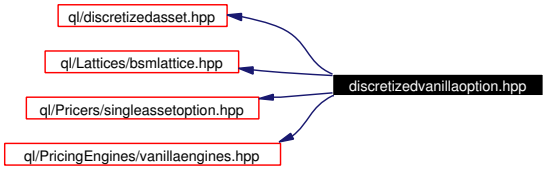
12.356 ql/PricingEngines/discretizedvanillaoption.hpp File Reference

12.356.1 Detailed Description

discretized vanilla option

```
#include <ql/discretizedasset.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Pricers/singleassetoption.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

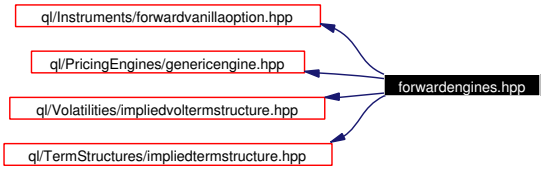
12.357 ql/PricingEngines/forwardengines.hpp File Reference

12.357.1 Detailed Description

Forward (strike-resetting) option engines.

```
#include <ql/Instruments/forwardvanillaoption.hpp>
#include <ql/PricingEngines/genericengine.hpp>
#include <ql/Volatilities/impliedvoltermstructure.hpp>
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Include dependency graph for forwardengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.358 ql/PricingEngines/genericengine.hpp File Reference

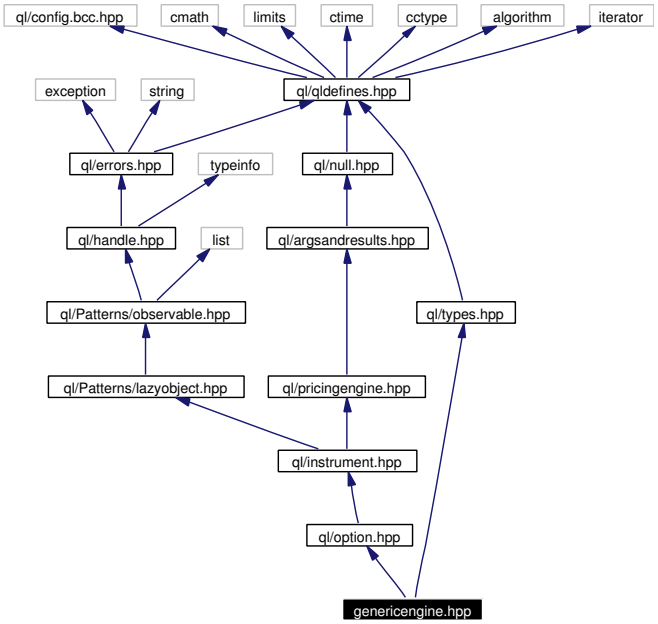
12.358.1 Detailed Description

Generic option engine.

`#include <ql/option.hpp>`

`#include <ql/types.hpp>`

Include dependency graph for genericengine.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.359 ql/PricingEngines/integralengines.cpp File Reference

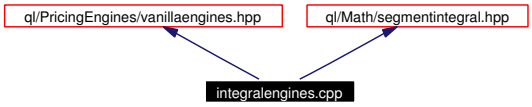
12.359.1 Detailed Description

Option engines using the integral approach.

`#include <ql/PricingEngines/vanillaengines.hpp>`

`#include <ql/Math/segmentintegral.hpp>`

Include dependency graph for integralengines.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.360 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

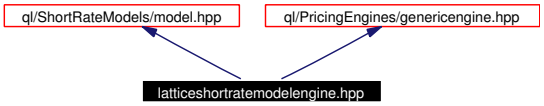
12.360.1 Detailed Description

Engine for a short-rate model specialized on a lattice.

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/PricingEngines/genericengine.hpp>`

Include dependency graph for latticeshortratemodelengine.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

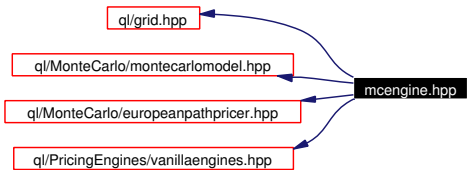
12.361 ql/PricingEngines/mcengine.hpp File Reference

12.361.1 Detailed Description

base class for Monte Carlo engines

```
#include <ql/grid.hpp>
#include <ql/MonteCarlo/montecarlomodel.hpp>
#include <ql/MonteCarlo/europeanpathpricer.hpp>
#include <ql/PricingEngines/vanillaengines.hpp>
```

Include dependency graph for mcengine.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.362 ql/PricingEngines/quantoengines.hpp File Reference

12.362.1 Detailed Description

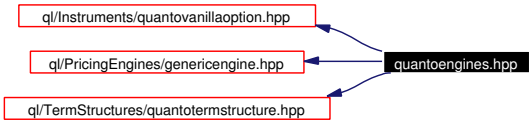
Quanto option engines.

`#include <ql/Instruments/quantovanillaoption.hpp>`

`#include <ql/PricingEngines/genericengine.hpp>`

`#include <ql/TermStructures/quantotermstructure.hpp>`

Include dependency graph for quantoengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

12.363 ql/PricingEngines/vanillaengines.hpp File Reference

12.363.1 Detailed Description

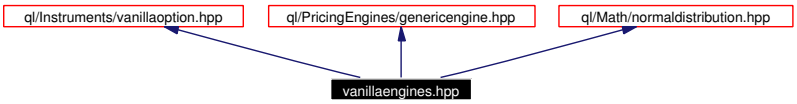
Vanilla option engines.

`#include <ql/Instruments/vanillaoption.hpp>`

`#include <ql/PricingEngines/genericengine.hpp>`

`#include <ql/Math/normaldistribution.hpp>`

Include dependency graph for vanillaengines.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::PricingEngines**

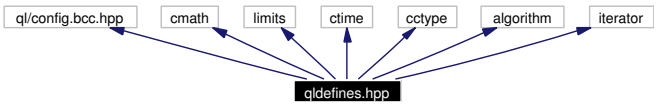
12.364 ql/qldefines.hpp File Reference

12.364.1 Detailed Description

Global definitions and compiler switches.

```
#include <ql/config.bcc.hpp>
#include <cmath>
#include <limits>
#include <ctime>
#include <cctype>
#include <algorithm>
#include <iterator>
```

Include dependency graph for qldefines.hpp:



Namespaces

- namespace **QuantLib**

Defines

- #define **QL_HEX_VERSION** 0x000304f0
  version hexadecimal number
- #define **QL_VERSION** "0.3.4"
  version string
- #define **QL_DUMMY_RETURN**(x)
  specific per-compiler definitions Is a dummy return statement required?
- #define **QL_IO_INIT** std::cout << std::string();
  I/O initialization.
- #define **QL_MIN_INT** ((std::numeric_limits<int>::min)())
- #define **QL_MAX_INT** ((std::numeric_limits<int>::max)())
- #define **QL_MIN_DOUBLE** -((std::numeric_limits<double>::max)())
- #define **QL_MAX_DOUBLE** ((std::numeric_limits<double>::max)())
- #define **QL_EPSILON** ((std::numeric_limits<double>::epsilon)())
- #define **QL_MIN_POSITIVE_DOUBLE** ((std::numeric_limits<double>::min)())
- #define **QL_DECLARE_TEMPLATE_SPECIALIZATIONS**
  Blame Microsoft for this one...

- #define **QL_ALLOW_TEMPLATE_METHOD_CALLS** 1

  Blame Microsoft for this one...

- #define **QL_TYPENAME** typename

  Blame Microsoft for this one...

- #define **QL_TEMPLATE_METAPROGRAMMING_WORKS** 1
- #define **QL_SPECIALIZE_ITERATOR_TRAITS**(T)
- #define **QL_REVERSE_ITERATOR**(iterator, type) std::reverse_iterator< iterator >

  Blame Microsoft for this one...

- #define **QL_FULL_ITERATOR_SUPPORT**

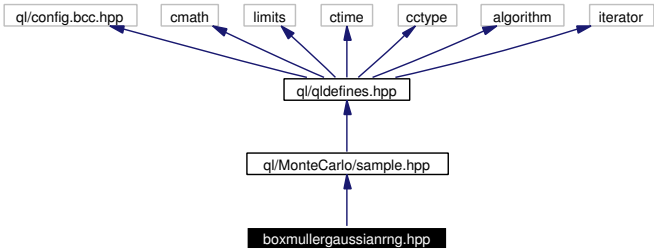
12.365 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

12.365.1 Detailed Description

Box-Muller Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for boxmullergaussianrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

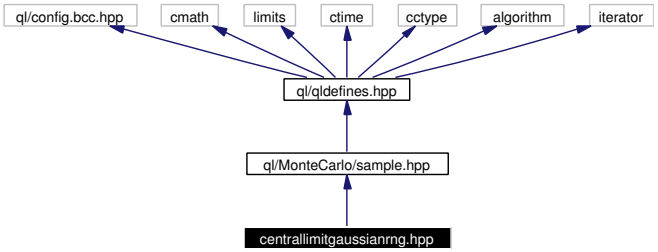
12.366 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

12.366.1 Detailed Description

Central limit Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for centrallimitgaussianrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

12.367 ql/RandomNumbers/haltonrsg.cpp File Reference

12.367.1 Detailed Description

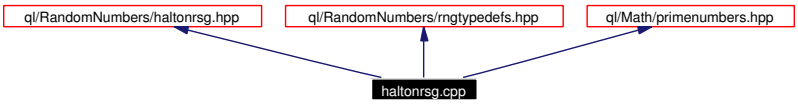
Halton low-discrepancy sequence generator.

`#include <ql/RandomNumbers/haltonrsg.hpp>`

`#include <ql/RandomNumbers/rngtypedefs.hpp>`

`#include <ql/Math/primenumbers.hpp>`

Include dependency graph for haltonrsg.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

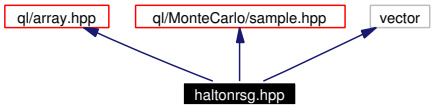
12.368 ql/RandomNumbers/haltonrsg.hpp File Reference

12.368.1 Detailed Description

Halton low-discrepancy sequence generator.

```
#include <ql/array.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <vector>
```

Include dependency graph for haltonrsg.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

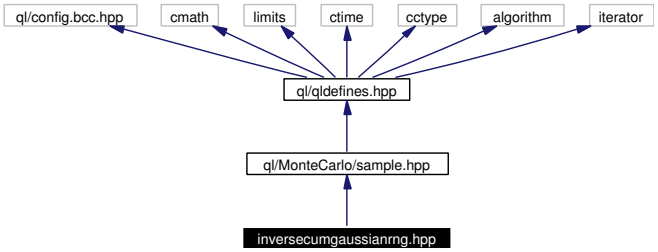
12.369 ql/RandomNumbers/inversecumgaussianrng.hpp File Reference

12.369.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for inversecumgaussianrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

12.370 ql/RandomNumbers/inversecumgaussianrsg.hpp File Reference

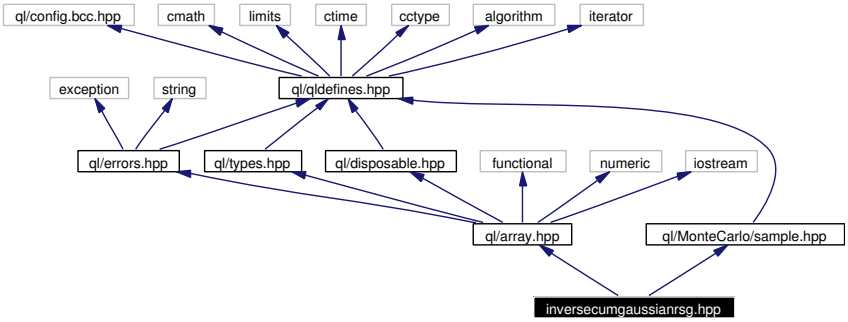
12.370.1 Detailed Description

Inverse cumulative Gaussian random sequence generator.

`#include <ql/array.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for inversecumgaussianrsg.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

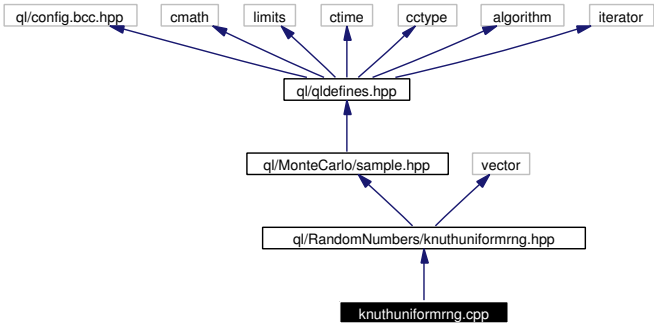
12.371 ql/RandomNumbers/knuthuniformrng.cpp File Reference

12.371.1 Detailed Description

Knuth uniform random number generator.

`#include <ql/RandomNumbers/knuthuniformrng.hpp>`

Include dependency graph for knuthuniformrng.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

12.372 ql/RandomNumbers/knuthuniformrng.hpp File Reference

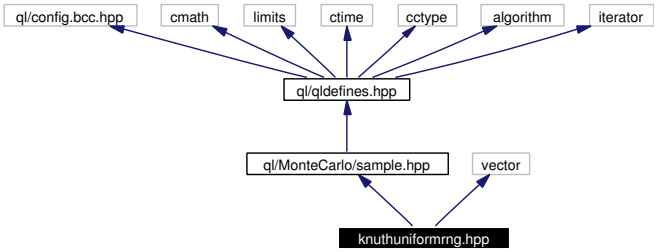
12.372.1 Detailed Description

Knuth uniform random number generator.

`#include <ql/MonteCarlo/sample.hpp>`

`#include <vector>`

Include dependency graph for knuthuniformrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

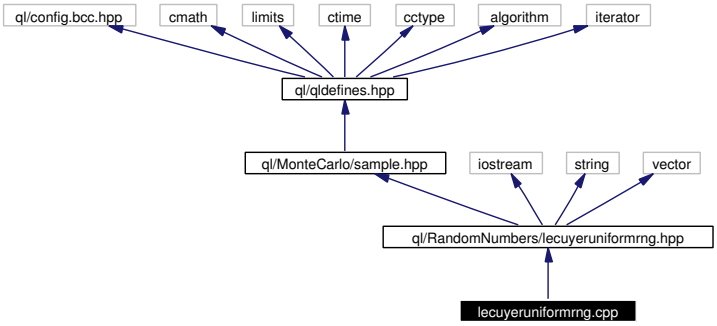
12.373 ql/RandomNumbers/lecuyeruniformrng.cpp File Reference

12.373.1 Detailed Description

L'Ecuyer uniform random number generator.

`#include <ql/RandomNumbers/lecuyeruniformrng.hpp>`

Include dependency graph for lecuyeruniformrng.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

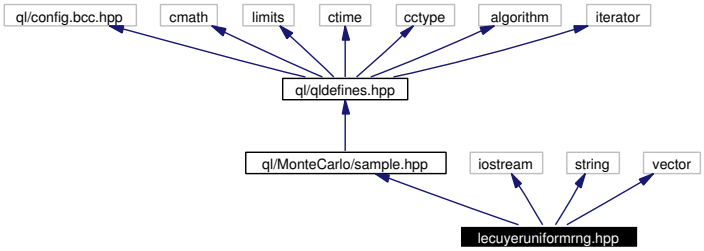
12.374 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

12.374.1 Detailed Description

L'Ecuyer uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
#include <iostream>
#include <string>
#include <vector>
```

Include dependency graph for lecuyeruniformrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

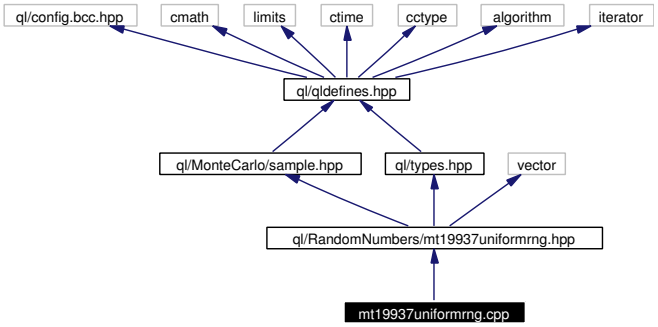
12.375 ql/RandomNumbers/mt19937uniformrng.cpp File Reference

12.375.1 Detailed Description

Mersenne Twister uniform random number generator.

`#include <ql/RandomNumbers/mt19937uniformrng.hpp>`

Include dependency graph for mt19937uniformrng.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

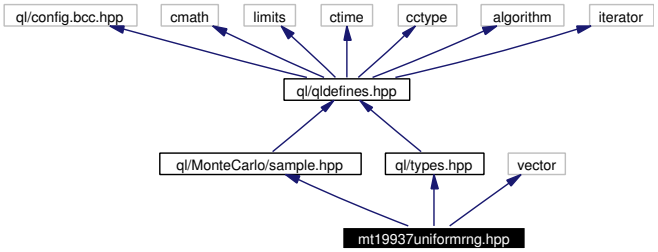
12.376 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

12.376.1 Detailed Description

Mersenne Twister uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
#include <ql/types.hpp>
#include <vector>
```

Include dependency graph for mt19937uniformrng.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

12.377 ql/RandomNumbers/randomarraygenerator.hpp File Reference

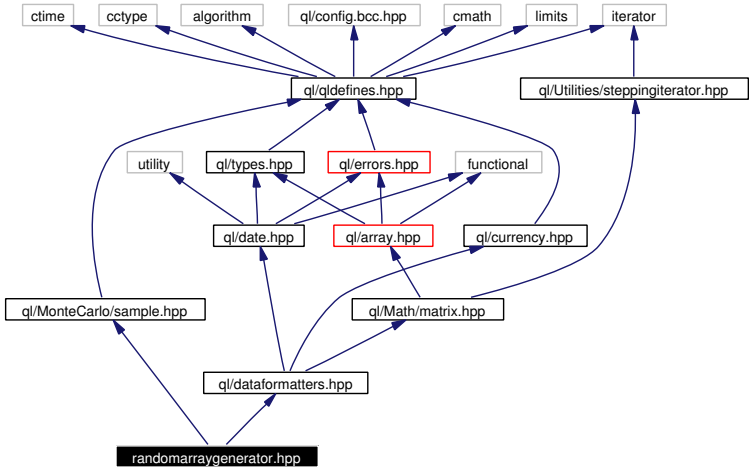
12.377.1 Detailed Description

Generates random arrays from a random number generator.

`#include <ql/MonteCarlo/sample.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for randomarraygenerator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

12.378 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

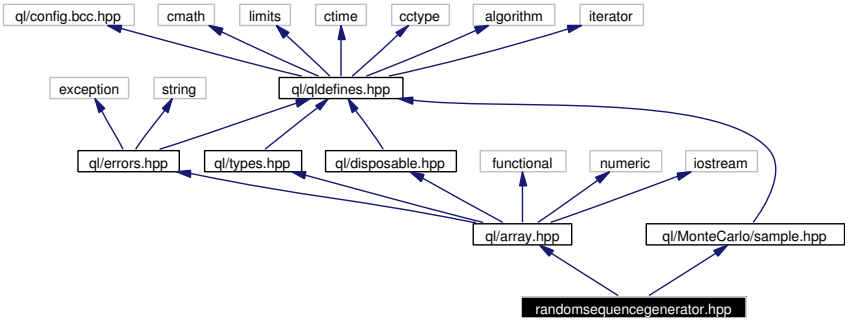
12.378.1 Detailed Description

Random sequence generator based on a pseudo-random number generator.

`#include <ql/array.hpp>`

`#include <ql/MonteCarlo/sample.hpp>`

Include dependency graph for randomsequencegenerator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

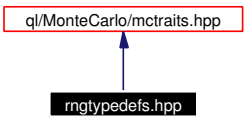
12.379 ql/RandomNumbers/rngtypedefs.hpp File Reference

12.379.1 Detailed Description

Default choices for template instantiations.

`#include <ql/MonteCarlo/mctraits.hpp>`

Include dependency graph for rngtypedefs.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

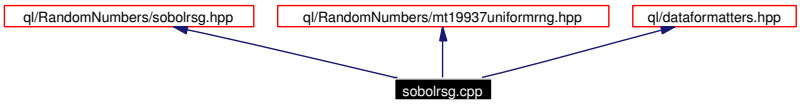
12.380 ql/RandomNumbers/sobolrsg.cpp File Reference

12.380.1 Detailed Description

Sobol low-discrepancy sequence generator.

```
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/primitivepolynomials.h>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/dataformatters.hpp>
```

Include dependency graph for sobolrsg.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

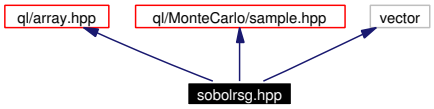
12.381 ql/RandomNumbers/sobolrsg.hpp File Reference

12.381.1 Detailed Description

Sobol low-discrepancy sequence generator.

```
#include <ql/array.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <vector>
```

Include dependency graph for sobolrsg.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::RandomNumbers**

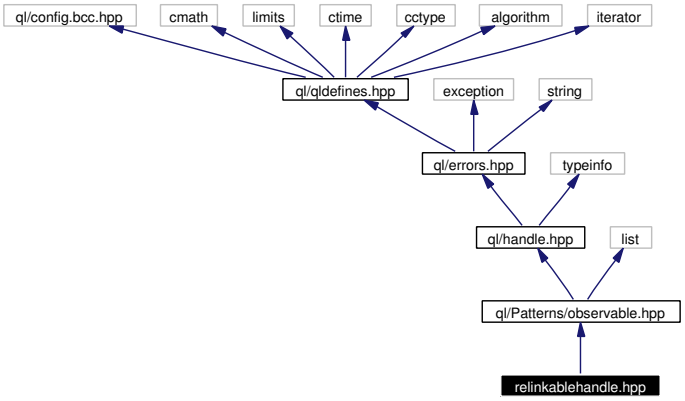
12.382 ql/relinkablehandle.hpp File Reference

12.382.1 Detailed Description

Globally accessible relinkable pointer.

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for relinkablehandle.hpp:



Namespaces

- namespace **QuantLib**

12.383 ql/scheduler.cpp File Reference

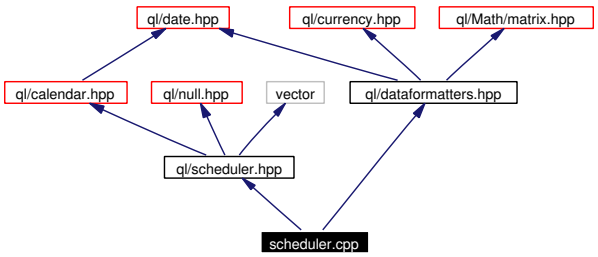
12.383.1 Detailed Description

date scheduler

`#include <ql/scheduler.hpp>`

`#include <ql/dataformatters.hpp>`

Include dependency graph for scheduler.cpp:



Namespaces

- namespace **QuantLib**

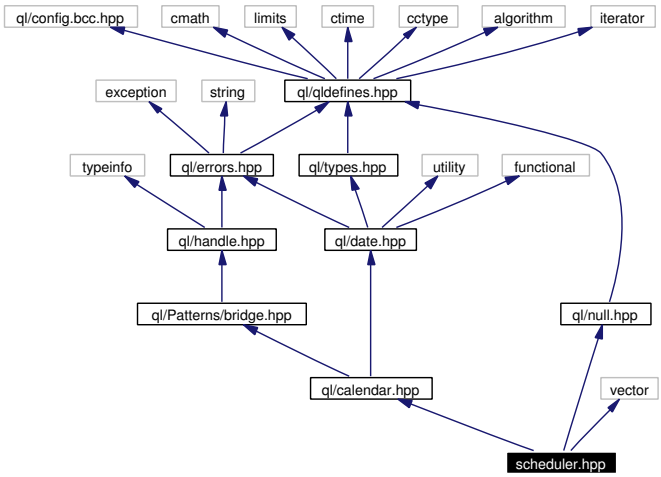
12.384 ql/scheduler.hpp File Reference

12.384.1 Detailed Description

date scheduler

```
#include <ql/calendar.hpp>
#include <ql/null.hpp>
#include <vector>
```

Include dependency graph for scheduler.hpp:



Namespaces

- namespace **QuantLib**

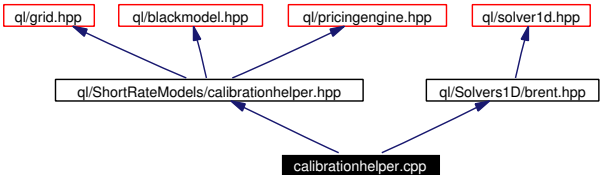
12.385 ql/ShortRateModels/calibrationhelper.cpp File Reference

12.385.1 Detailed Description

Calibration helper class.

```
#include "ql/ShortRateModels/calibrationhelper.hpp"
#include "ql/Solvers1D/brent.hpp"
```

Include dependency graph for calibrationhelper.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

12.386 ql/ShortRateModels/calibrationhelper.hpp File Reference

12.386.1 Detailed Description

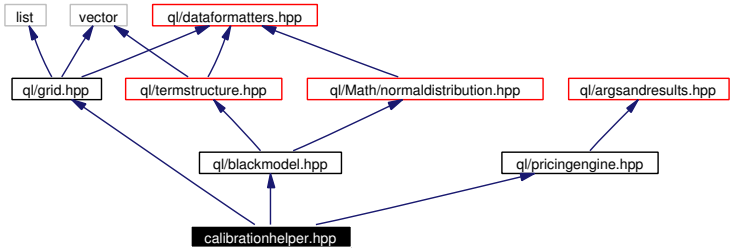
Calibration helper class.

`#include <ql/grid.hpp>`

`#include <ql/blackmodel.hpp>`

`#include <ql/pricingengine.hpp>`

Include dependency graph for calibrationhelper.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

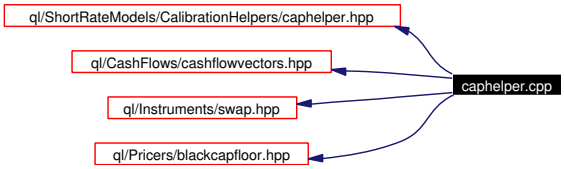
12.387 ql/ShortRateModels/CalibrationHelpers/caphelper.cpp File Reference

12.387.1 Detailed Description

Cap calibration helper.

```
#include "ql/ShortRateModels/CalibrationHelpers/caphelper.hpp"
#include "ql/CashFlows/cashflowvectors.hpp"
#include "ql/Instruments/swap.hpp"
#include "ql/Pricers/blackcapfloor.hpp"
```

Include dependency graph for caphelper.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**
- namespace **QuantLib::ShortRateModels::CalibrationHelpers**

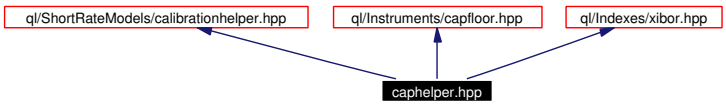
12.388 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

12.388.1 Detailed Description

CapHelper calibration helper.

```
#include "ql/ShortRateModels/calibrationhelper.hpp"
#include "ql/Instruments/capfloor.hpp"
#include "ql/Indexes/xibor.hpp"
```

Include dependency graph for caphelper.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**
- namespace **QuantLib::ShortRateModels::CalibrationHelpers**

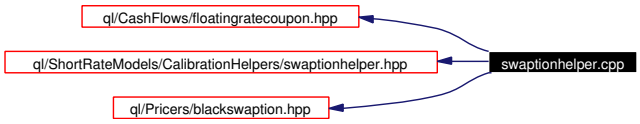
12.389 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.cpp File Reference

12.389.1 Detailed Description

Swaption calibration helper.

```
#include "ql/CashFlows/floatingratecoupon.hpp"
#include "ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp"
#include "ql/Pricers/blackswaption.hpp"
```

Include dependency graph for swaptionhelper.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**
- namespace **QuantLib::ShortRateModels::CalibrationHelpers**

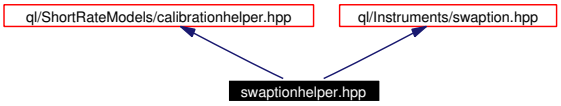
12.390 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

12.390.1 Detailed Description

Swaption calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for swaptionhelper.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**
- namespace **QuantLib::ShortRateModels::CalibrationHelpers**

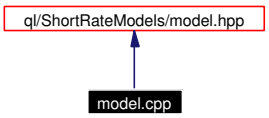
12.391 ql/ShortRateModels/model.cpp File Reference

12.391.1 Detailed Description

Abstract interest rate model class.

`#include "ql/ShortRateModels/model.hpp"`

Include dependency graph for model.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

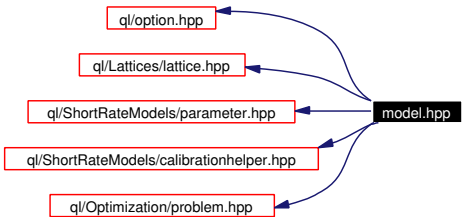
12.392 ql/ShortRateModels/model.hpp File Reference

12.392.1 Detailed Description

Abstract interest rate model class.

```
#include <ql/option.hpp>
#include <ql/Lattices/lattice.hpp>
#include <ql/ShortRateModels/parameter.hpp>
#include <ql/ShortRateModels/calibrationhelper.hpp>
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for model.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

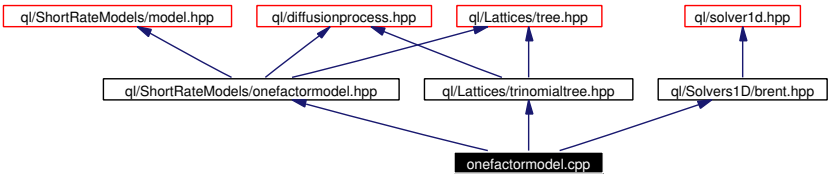
12.393 ql/ShortRateModels/onefactormodel.cpp File Reference

12.393.1 Detailed Description

Abstract one-factor interest rate model class.

```
#include "ql/ShortRateModels/onefactormodel.hpp"
#include "ql/Lattices/trinomialtree.hpp"
#include "ql/Solvers1D/brent.hpp"
```

Include dependency graph for onefactormodel.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

12.394 ql/ShortRateModels/onefactormodel.hpp File Reference

12.394.1 Detailed Description

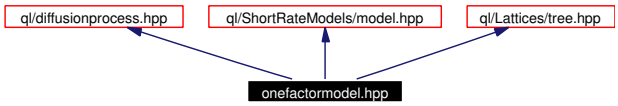
Abstract one-factor interest rate model class.

`#include <ql/diffusionprocess.hpp>`

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/Lattices/tree.hpp>`

Include dependency graph for onefactormodel.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

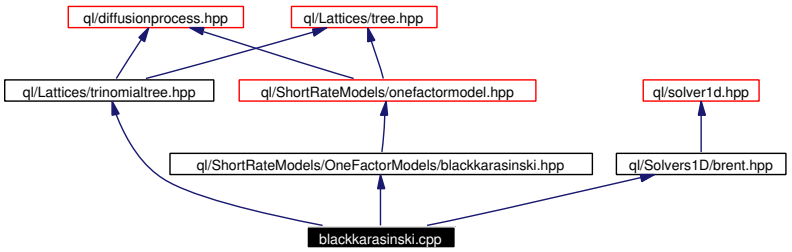
12.395 ql/ShortRateModels/OneFactorModels/blackkarasinski.cpp File Reference

12.395.1 Detailed Description

Black-Karasinski model.

```
#include "ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp"
#include "ql/Lattices/trinomialtree.hpp"
#include "ql/Solvers1D/brent.hpp"
```

Include dependency graph for blackkarasinski.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

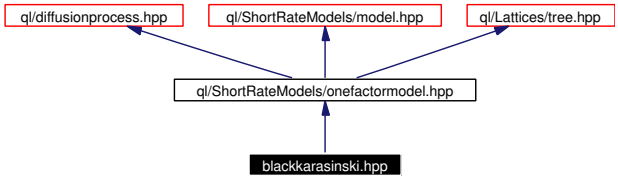
12.396 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

12.396.1 Detailed Description

Black-Karasinski model.

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for blackkarasinski.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

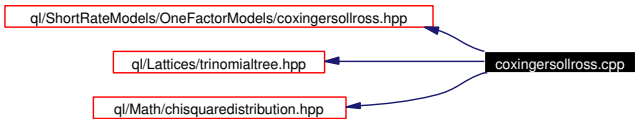
12.397 ql/ShortRateModels/OneFactorModels/coxingersollross.cpp File Reference

12.397.1 Detailed Description

Cox-Ingersoll-Ross model.

```
#include "ql/ShortRateModels/OneFactorModels/coxingersollross.hpp"
#include "ql/Lattices/trinomialtree.hpp"
#include "ql/Math/chisquaredistribution.hpp"
```

Include dependency graph for coxingersollross.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

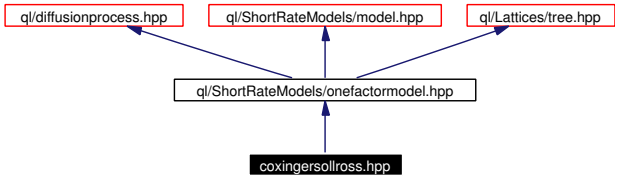
12.398 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

12.398.1 Detailed Description

Cox-Ingersoll-Ross model.

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for coxingersollross.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

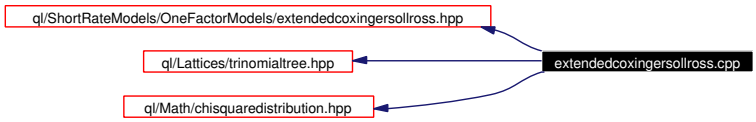
12.399 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.cpp File Reference

12.399.1 Detailed Description

extended Cox-Ingersoll-Ross model

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
#include "ql/Lattices/trinomialtree.hpp"
#include <ql/Math/chisquaredistribution.hpp>
```

Include dependency graph for extendedcoxingersollross.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

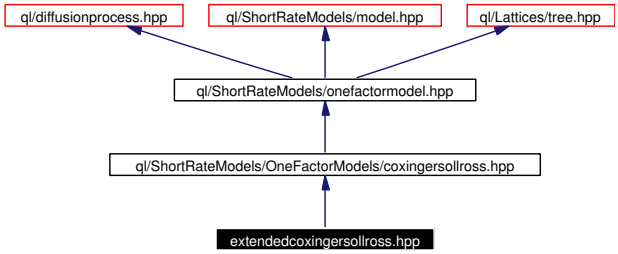
12.400 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

12.400.1 Detailed Description

Extended Cox-Ingersoll-Ross model.

`#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>`

Include dependency graph for extendedcoxingersollross.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

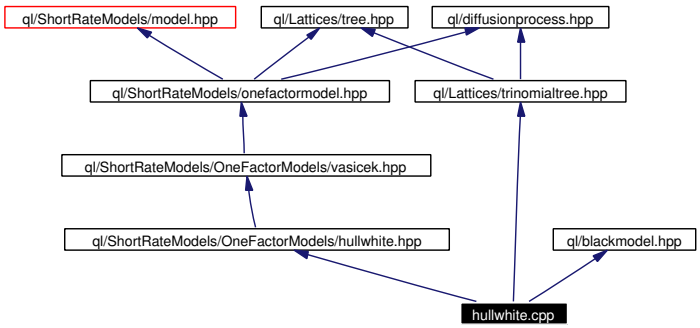
12.401 ql/ShortRateModels/OneFactorModels/hullwhite.cpp File Reference

12.401.1 Detailed Description

Hull & White model.

```
#include "ql/ShortRateModels/OneFactorModels/hullwhite.hpp"
#include "ql/Lattices/trinomialtree.hpp"
#include "ql/blackmodel.hpp"
```

Include dependency graph for hullwhite.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

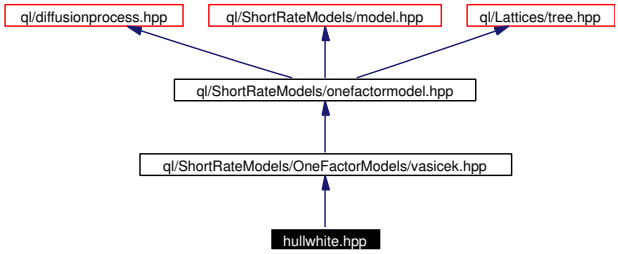
12.402 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

12.402.1 Detailed Description

Hull & White (HW) model.

`#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>`

Include dependency graph for hullwhite.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

12.403 ql/ShortRateModels/OneFactorModels/vasicek.cpp File Reference

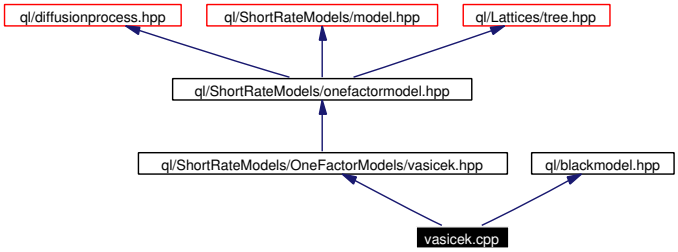
12.403.1 Detailed Description

Vasicek model class.

`#include "ql/ShortRateModels/OneFactorModels/vasicek.hpp"`

`#include "ql/blackmodel.hpp"`

Include dependency graph for vasicek.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

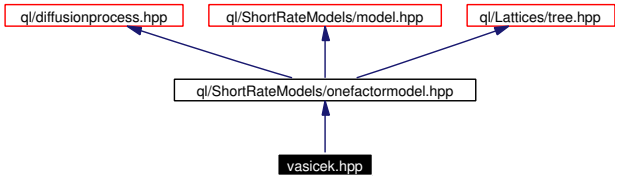
12.404 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

12.404.1 Detailed Description

Vasicek model class.

`#include <ql/ShortRateModels/onefactormodel.hpp>`

Include dependency graph for vasicek.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

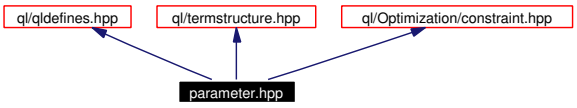
12.405 ql/ShortRateModels/parameter.hpp File Reference

12.405.1 Detailed Description

Model parameter classes.

```
#include <ql/qldefines.hpp>
#include <ql/termstructure.hpp>
#include <ql/Optimization/constraint.hpp>
```

Include dependency graph for parameter.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

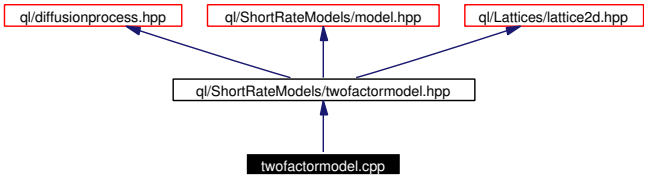
12.406 ql/ShortRateModels/twofactormodel.cpp File Reference

12.406.1 Detailed Description

Abstract two-factor interest rate model class.

`#include <ql/ShortRateModels/twofactormodel.hpp>`

Include dependency graph for twofactormodel.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

12.407 ql/ShortRateModels/twofactormodel.hpp File Reference

12.407.1 Detailed Description

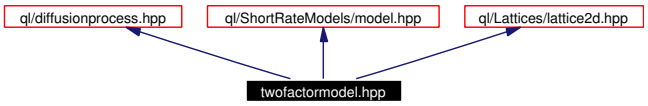
Abstract two-factor interest rate model class.

`#include <ql/diffusionprocess.hpp>`

`#include <ql/ShortRateModels/model.hpp>`

`#include <ql/Lattices/lattice2d.hpp>`

Include dependency graph for twofactormodel.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

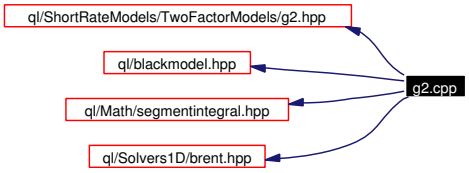
12.408 ql/ShortRateModels/TwoFactorModels/g2.cpp File Reference

12.408.1 Detailed Description

Two-factor additive Gaussian Model G2++.

```
#include "ql/ShortRateModels/TwoFactorModels/g2.hpp"
#include "ql/blackmodel.hpp"
#include "ql/Math/segmentintegral.hpp"
#include "ql/Solvers1D/brent.hpp"
```

Include dependency graph for g2.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

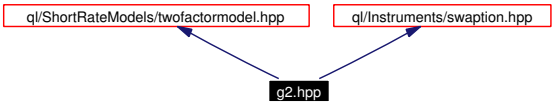
12.409 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

12.409.1 Detailed Description

Two-factor additive Gaussian Model G2++.

```
#include <ql/ShortRateModels/twofactormodel.hpp>
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for g2.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::ShortRateModels**

12.410 ql/solver1d.hpp File Reference

12.410.1 Detailed Description

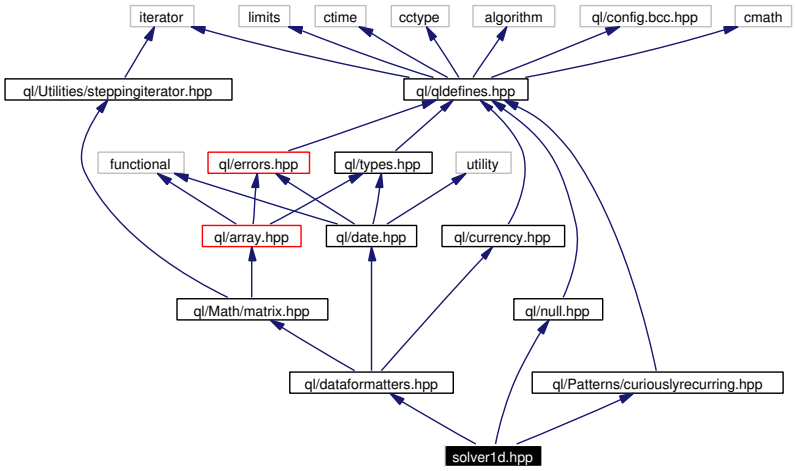
Abstract 1-D solver class.

`#include <ql/null.hpp>`

`#include <ql/dataformatters.hpp>`

`#include <ql/Patterns/curiouslyrecurring.hpp>`

Include dependency graph for solver1d.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

Defines

- #define **MAX_FUNCTION_EVALUATIONS** 100

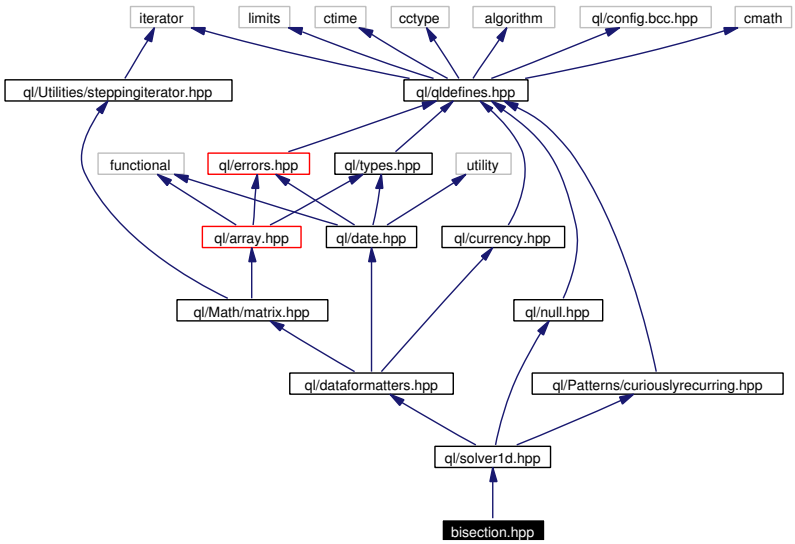
12.411 ql/Solvers1D/bisection.hpp File Reference

12.411.1 Detailed Description

bisection 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for bisection.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

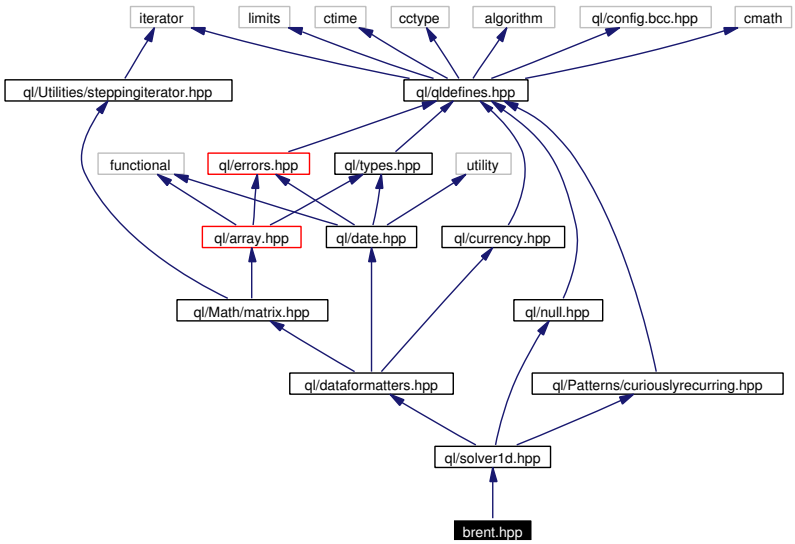
12.412 ql/Solvers1D/brent.hpp File Reference

12.412.1 Detailed Description

Brent 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for brent.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

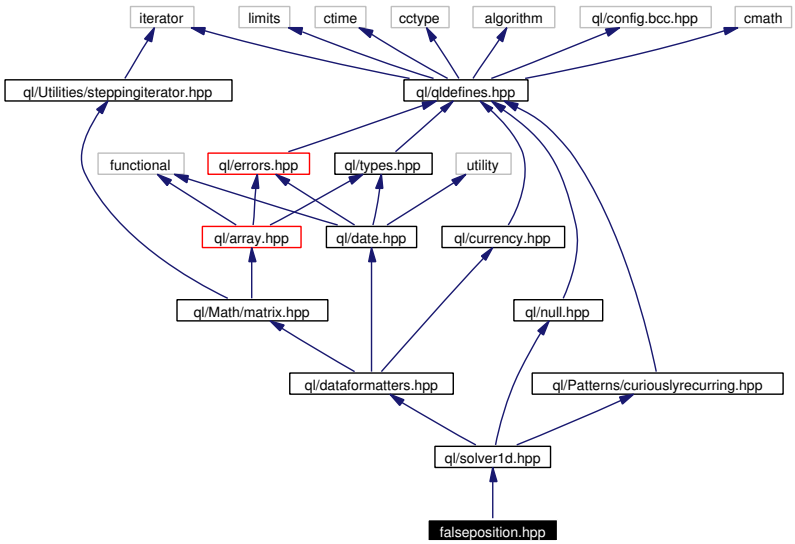
12.413 ql/Solvers1D/falseposition.hpp File Reference

12.413.1 Detailed Description

false-position 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for falseposition.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

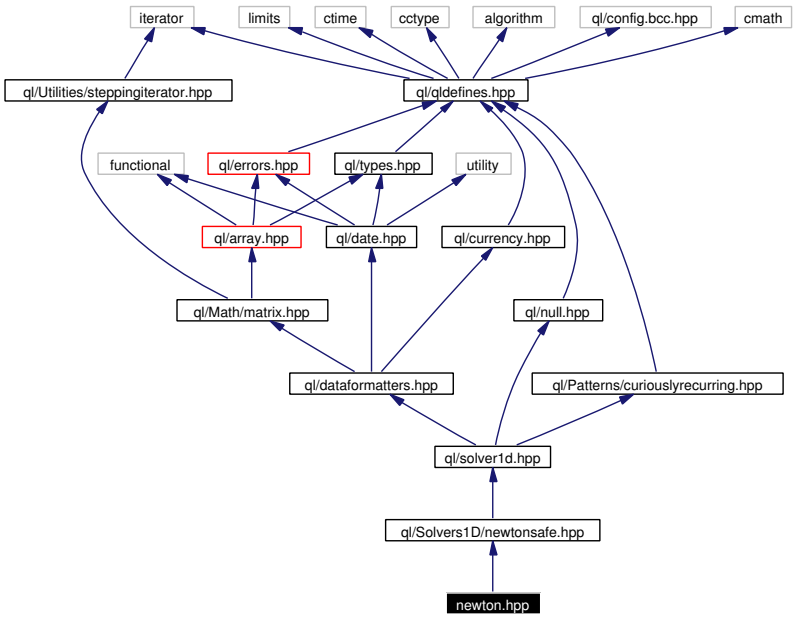
12.414 ql/Solvers1D/newton.hpp File Reference

12.414.1 Detailed Description

Newton 1-D solver.

`#include <ql/Solvers1D/newtonsafe.hpp>`

Include dependency graph for newton.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

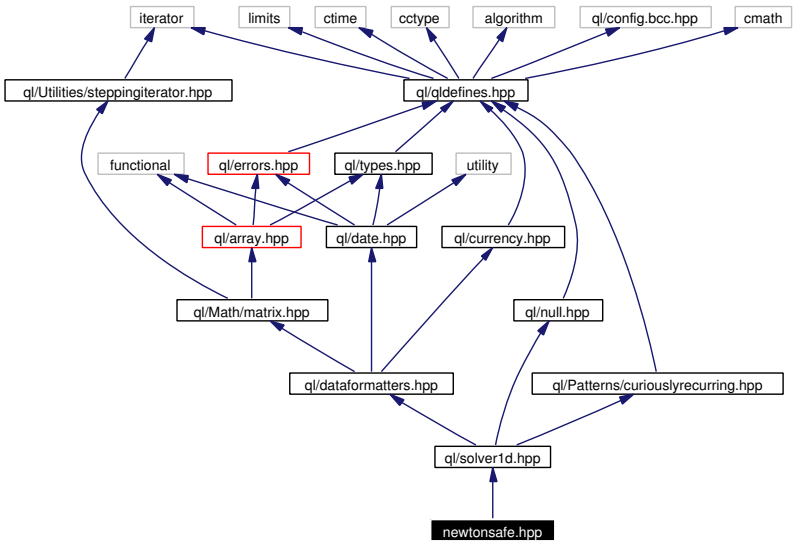
12.415 ql/Solvers1D/newtonsafe.hpp File Reference

12.415.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for newtonsafe.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

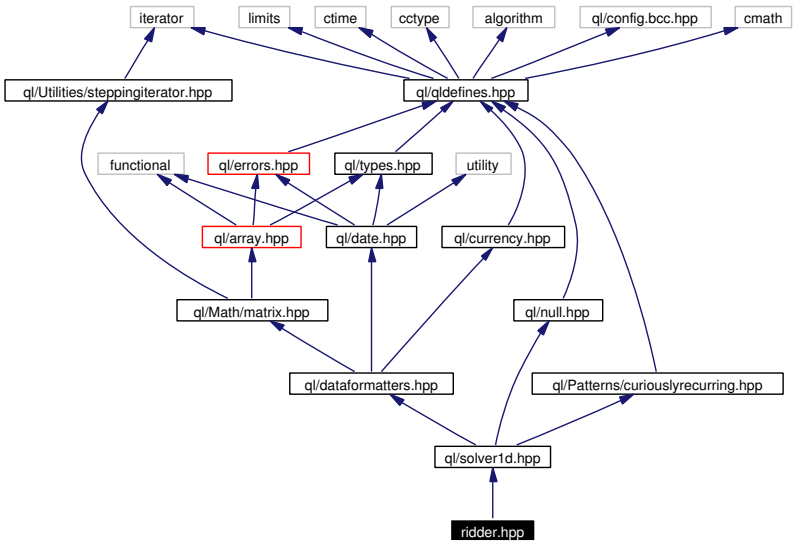
12.416 ql/Solvers1D/ridder.hpp File Reference

12.416.1 Detailed Description

Ridder 1-D solver.

`#include <ql/solver1d.hpp>`

Include dependency graph for ridder.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

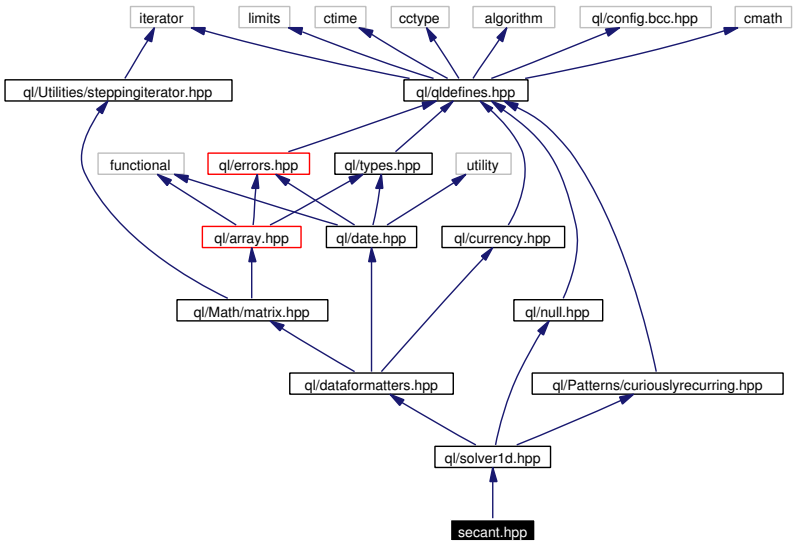
12.417 ql/Solvers1D/secant.hpp File Reference

12.417.1 Detailed Description

secant 1-D solver

`#include <ql/solver1d.hpp>`

Include dependency graph for secant.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Solvers1D**

12.418 ql/swaptionvolstructure.hpp File Reference

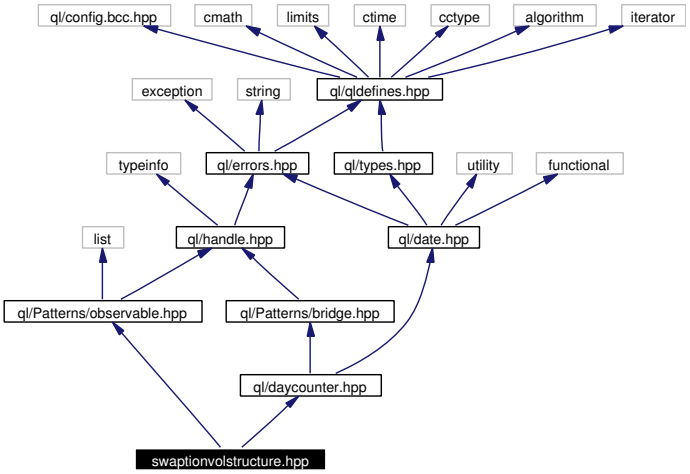
12.418.1 Detailed Description

Swaption volatility structure.

`#include <ql/daycounter.hpp>`

`#include <ql/Patterns/observable.hpp>`

Include dependency graph for swaptionvolstructure.hpp:



Namespaces

- namespace **QuantLib**

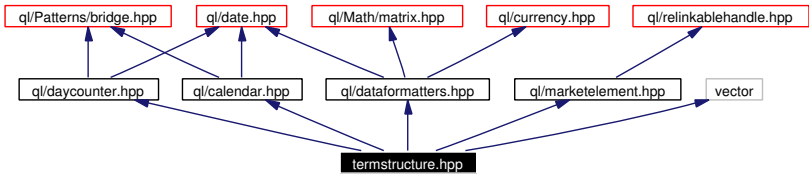
12.419 ql/termstructure.hpp File Reference

12.419.1 Detailed Description

Term structure.

```
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/dataformatters.hpp>
#include <ql/marketelement.hpp>
#include <vector>
```

Include dependency graph for termstructure.hpp:



Namespaces

- namespace **QuantLib::TermStructures**
- namespace **QuantLib**

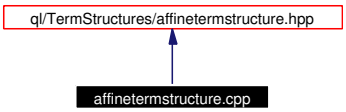
12.420 ql/TermStructures/affinetermstructure.cpp File Reference

12.420.1 Detailed Description

Affine term-structure.

`#include <ql/TermStructures/affinetermstructure.hpp>`

Include dependency graph for affinetermstructure.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

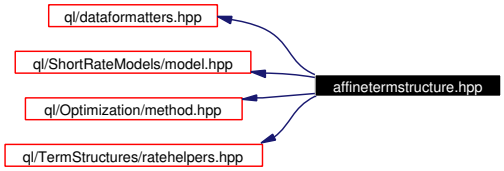
12.421 ql/TermStructures/affinetermstructure.hpp File Reference

12.421.1 Detailed Description

Affine term structure.

```
#include <ql/dataformatters.hpp>
#include <ql/ShortRateModels/model.hpp>
#include <ql/Optimization/method.hpp>
#include <ql/TermStructures/ratehelpers.hpp>
```

Include dependency graph for affinetermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.422 ql/TermStructures/compoundforward.cpp File Reference

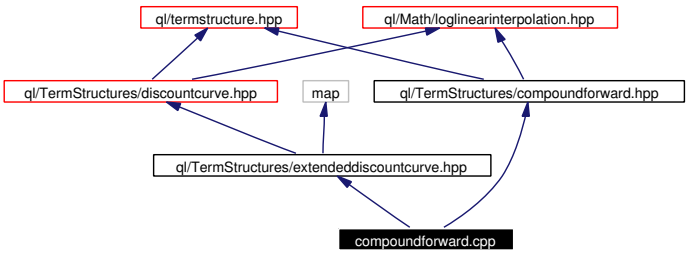
12.422.1 Detailed Description

compounded forward term structure

`#include <ql/TermStructures/compoundforward.hpp>`

`#include <ql/TermStructures/extendeddiscountcurve.hpp>`

Include dependency graph for compoundforward.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.423 ql/TermStructures/compoundforward.hpp File Reference

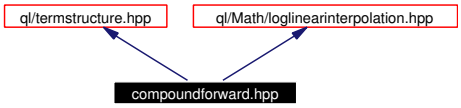
12.423.1 Detailed Description

compounded forward term structure

`#include <ql/termstructure.hpp>`

`#include <ql/Math/loglinearinterpolation.hpp>`

Include dependency graph for compoundforward.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.424 ql/TermStructures/discountcurve.cpp File Reference

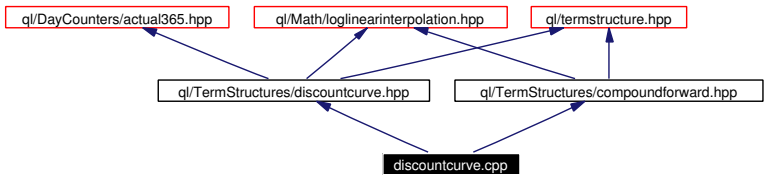
12.424.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

`#include <ql/TermStructures/discountcurve.hpp>`

`#include <ql/TermStructures/compoundforward.hpp>`

Include dependency graph for discountcurve.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.425 ql/TermStructures/discountcurve.hpp File Reference

12.425.1 Detailed Description

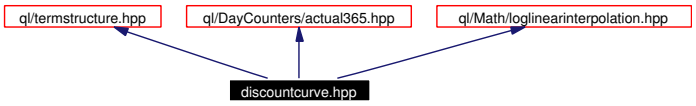
pre-bootstrapped discount factor structure

`#include <ql/termstructure.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

`#include <ql/Math/loglinearinterpolation.hpp>`

Include dependency graph for discountcurve.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.426 ql/TermStructures/drifttermstructure.hpp File Reference

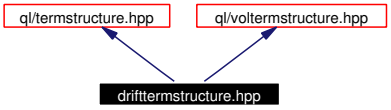
12.426.1 Detailed Description

Drift term structure.

`#include <ql/termstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for drifttermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.427 ql/TermStructures/extendeddiscountcurve.cpp File Reference

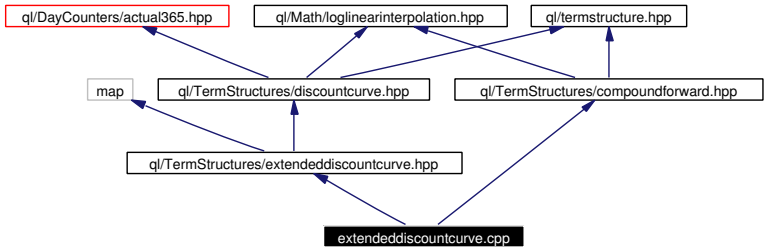
12.427.1 Detailed Description

discount factor structure with detailed compound-forward calculation

`#include <ql/TermStructures/extendeddiscountcurve.hpp>`

`#include <ql/TermStructures/compoundforward.hpp>`

Include dependency graph for extendeddiscountcurve.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.428 ql/TermStructures/extendeddiscountcurve.hpp File Reference

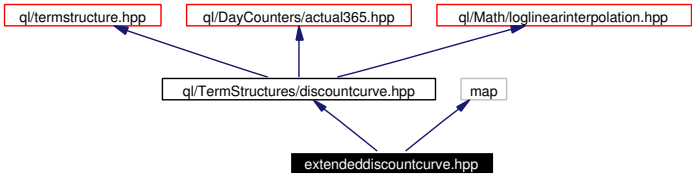
12.428.1 Detailed Description

discount factor structure with detailed compound-forward calculation

`#include <ql/TermStructures/discountcurve.hpp>`

`#include <map>`

Include dependency graph for extendeddiscountcurve.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

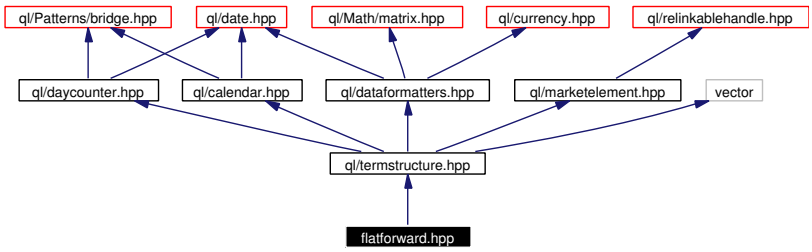
12.429 ql/TermStructures/flatforward.hpp File Reference

12.429.1 Detailed Description

flat forward rate term structure

`#include <ql/termstructure.hpp>`

Include dependency graph for flatforward.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

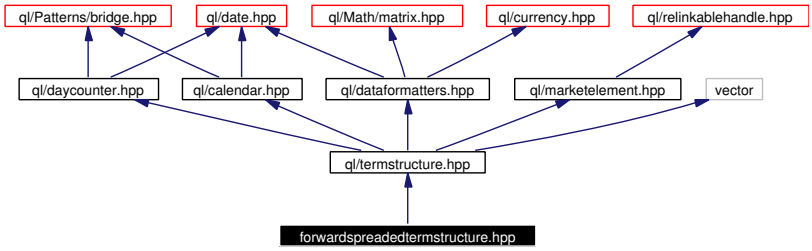
12.430 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

12.430.1 Detailed Description

Forward spreaded term structure.

`#include <ql/termstructure.hpp>`

Include dependency graph for forwardspreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

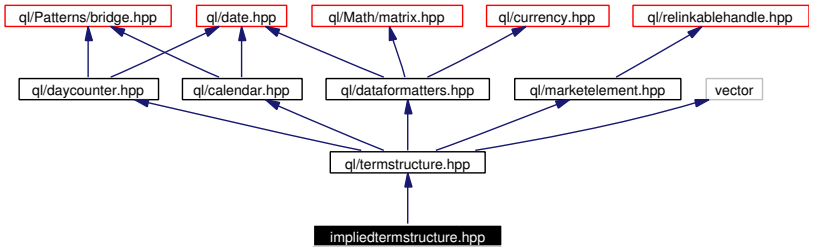
12.431 ql/TermStructures/impliedtermstructure.hpp File Reference

12.431.1 Detailed Description

Implied term structure.

`#include <ql/termstructure.hpp>`

Include dependency graph for impliedtermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.432 ql/TermStructures/piecewiseflatforward.cpp File Reference

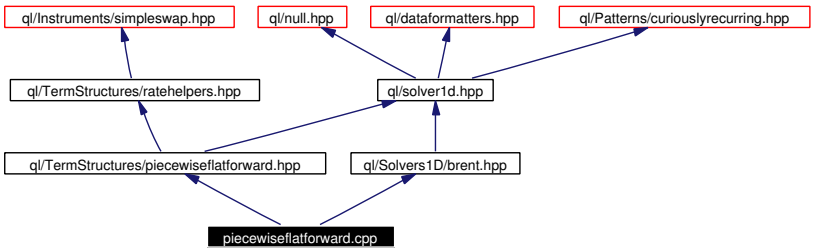
12.432.1 Detailed Description

piecewise flat forward term structure

`#include <ql/TermStructures/piecewiseflatforward.hpp>`

`#include <ql/Solvers1D/brent.hpp>`

Include dependency graph for piecewiseflatforward.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.433 ql/TermStructures/piecewiseflatforward.hpp File Reference

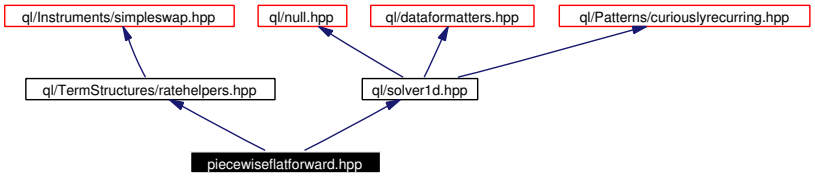
12.433.1 Detailed Description

piecewise flat forward term structure

`#include <ql/TermStructures/ratehelpers.hpp>`

`#include <ql/solver1d.hpp>`

Include dependency graph for piecewiseflatforward.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.434 ql/TermStructures/quantotermstructure.hpp File Reference

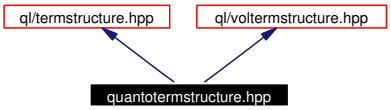
12.434.1 Detailed Description

Quanto term structure.

`#include <ql/termstructure.hpp>`

`#include <ql/voltermstructure.hpp>`

Include dependency graph for quantotermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

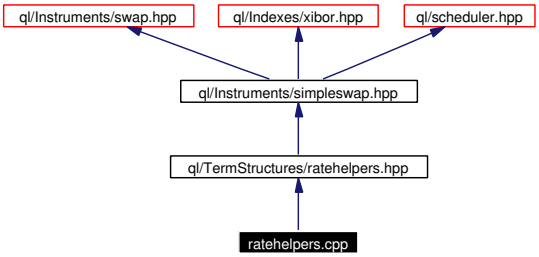
12.435 ql/TermStructures/ratehelpers.cpp File Reference

12.435.1 Detailed Description

rate helpers base class

`#include <ql/TermStructures/ratehelpers.hpp>`

Include dependency graph for ratehelpers.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

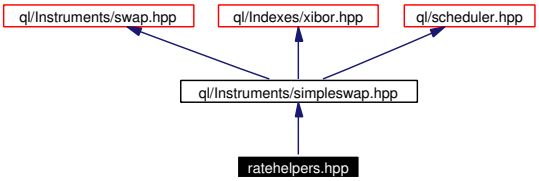
12.436 ql/TermStructures/ratehelpers.hpp File Reference

12.436.1 Detailed Description

rate helpers base class

`#include <ql/Instruments/simpleswap.hpp>`

Include dependency graph for ratehelpers.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

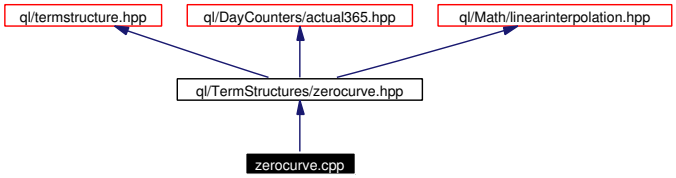
12.437 ql/TermStructures/zerocurve.cpp File Reference

12.437.1 Detailed Description

Term structure based on linear interpolation of zero yields.

`#include <ql/TermStructures/zerocurve.hpp>`

Include dependency graph for zerocurve.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

12.438 ql/TermStructures/zerocurve.hpp File Reference

12.438.1 Detailed Description

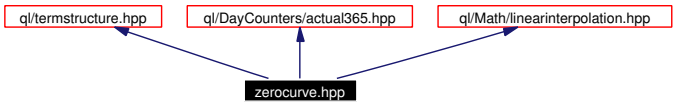
pre-bootstrapped zero curve structure

`#include <ql/termstructure.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

`#include <ql/Math/linearinterpolation.hpp>`

Include dependency graph for zerocurve.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

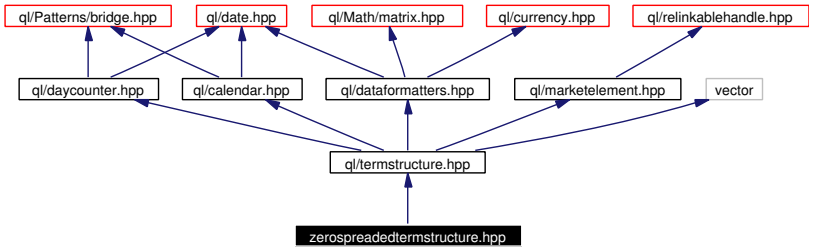
12.439 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

12.439.1 Detailed Description

Zero spreaded term structure.

`#include <ql/termstructure.hpp>`

Include dependency graph for zerospreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::TermStructures**

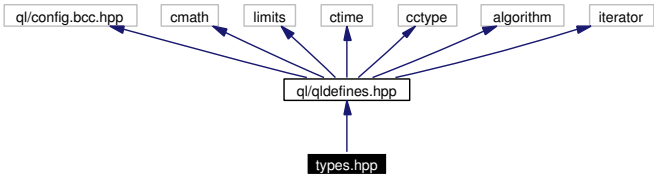
12.440 ql/types.hpp File Reference

12.440.1 Detailed Description

Custom types.

`#include <ql/qldefines.hpp>`

Include dependency graph for types.hpp:



Namespaces

- namespace **QuantLib**

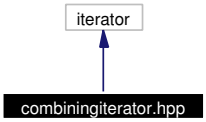
12.441 ql/Utilities/combiningiterator.hpp File Reference

12.441.1 Detailed Description

Iterator mapping a function to a set of underlying sequences.

`#include <iterator>`

Include dependency graph for combiningiterator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

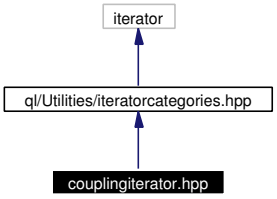
12.442 ql/Utilities/couplingiterator.hpp File Reference

12.442.1 Detailed Description

Iterator mapping a function to a pair of underlying sequences.

`#include <ql/Utilities/iteratorcategories.hpp>`

Include dependency graph for couplingiterator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

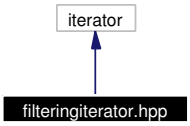
12.443 ql/Utilities/filteringiterator.hpp File Reference

12.443.1 Detailed Description

Iterator filtering undesired data.

`#include <iterator>`

Include dependency graph for filteringiterator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

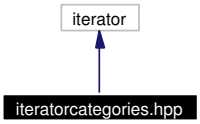
12.444 ql/Utilities/iteratorcategories.hpp File Reference

12.444.1 Detailed Description

Lowest common denominator between two iterator categories.

`#include <iterator>`

Include dependency graph for iteratorcategories.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

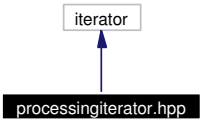
12.445 ql/Utilities/processingiterator.hpp File Reference

12.445.1 Detailed Description

Iterator mapping a unary function to an underlying sequence.

`#include <iterator>`

Include dependency graph for processingiterator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

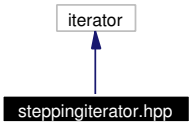
12.446 ql/Utilities/steppingiterator.hpp File Reference

12.446.1 Detailed Description

Iterator advancing in constant steps.

`#include <iterator>`

Include dependency graph for steppingiterator.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Utilities**

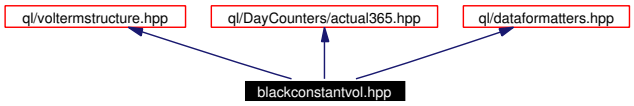
12.447 ql/Volatilities/blackconstantvol.hpp File Reference

12.447.1 Detailed Description

Black constant volatility, no time dependence, no strike dependence.

```
#include <ql/voltermstructure.hpp>
#include <ql/DayCounters/actual365.hpp>
#include <ql/dataformatters.hpp>
```

Include dependency graph for blackconstantvol.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

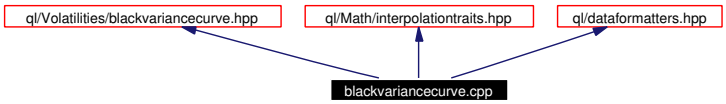
12.448 ql/Volatilities/blackvariancecurve.cpp File Reference

12.448.1 Detailed Description

Black volatility curve modelled as variance curve.

```
#include <ql/Volatilities/blackvariancecurve.hpp>
#include <ql/Math/interpolationtraits.hpp>
#include <ql/dataformatters.hpp>
```

Include dependency graph for blackvariancecurve.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

12.449 ql/Volatilities/blackvariancecurve.hpp File Reference

12.449.1 Detailed Description

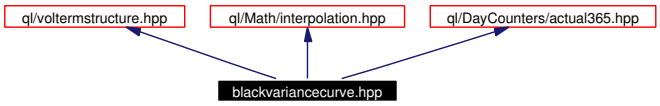
Black volatility curve modelled as variance curve.

`#include <ql/voltermstructure.hpp>`

`#include <ql/Math/interpolation.hpp>`

`#include <ql/DayCounters/actual365.hpp>`

Include dependency graph for blackvariancecurve.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

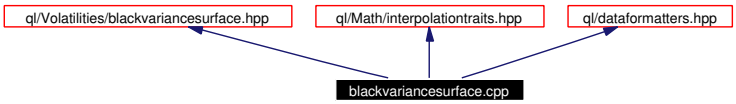
12.450 ql/Volatilities/blackvariancesurface.cpp File Reference

12.450.1 Detailed Description

Black volatility surface modelled as variance surface.

```
#include <ql/Volatilities/blackvariancesurface.hpp>
#include <ql/Math/interpolationtraits.hpp>
#include <ql/dataformatters.hpp>
```

Include dependency graph for blackvariancesurface.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

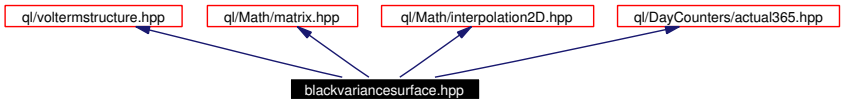
12.451 ql/Volatilities/blackvariancesurface.hpp File Reference

12.451.1 Detailed Description

Black volatility surface modelled as variance surface.

```
#include <ql/voltermstructure.hpp>
#include <ql/Math/matrix.hpp>
#include <ql/Math/interpolation2D.hpp>
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for blackvariancesurface.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

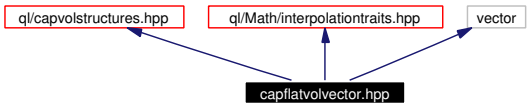
12.452 ql/Volatilities/capflatvolvector.hpp File Reference

12.452.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

```
#include "ql/capvolstructures.hpp"
#include "ql/Math/interpolationtraits.hpp"
#include <vector>
```

Include dependency graph for capflatvolvector.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Volatilities**

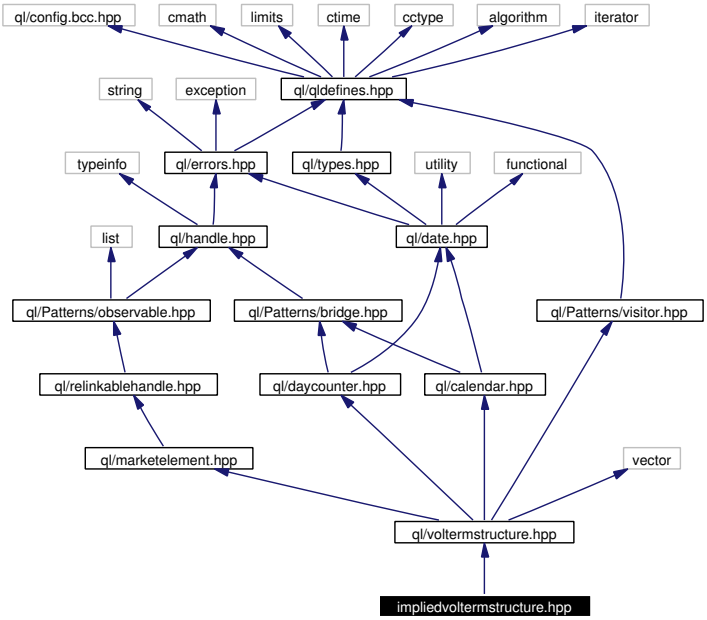
12.453 ql/Volatilities/impliedvoltermstructure.hpp File Reference

12.453.1 Detailed Description

Implied Black Vol Term Structure.

`#include <ql/voltermstructure.hpp>`

Include dependency graph for impliedvoltermstructure.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

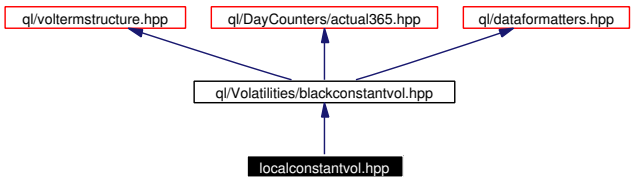
12.454 ql/Volatilities/localconstantvol.hpp File Reference

12.454.1 Detailed Description

Local constant volatility, no time dependence, no asset dependence.

`#include <ql/Volatilities/blackconstantvol.hpp>`

Include dependency graph for localconstantvol.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

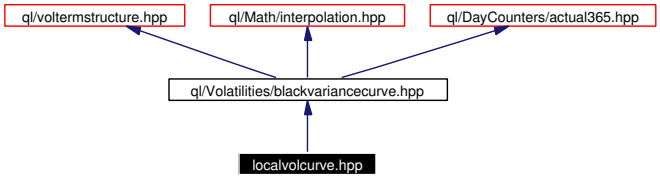
12.455 ql/Volatilities/localvolcurve.hpp File Reference

12.455.1 Detailed Description

Local volatility curve derived from a Black curve.

`#include <ql/Volatilities/blackvariancecurve.hpp>`

Include dependency graph for localvolcurve.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

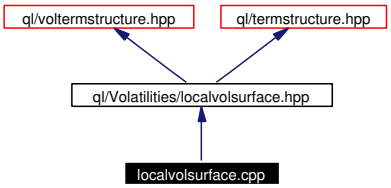
12.456 ql/Volatilities/localvolsurface.cpp File Reference

12.456.1 Detailed Description

Local volatility surface derived from a Black vol surface.

`#include <ql/Volatilities/localvolsurface.hpp>`

Include dependency graph for localvolsurface.cpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

12.457 ql/Volatilities/localvolsurface.hpp File Reference

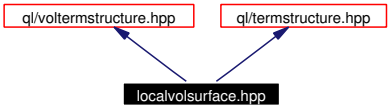
12.457.1 Detailed Description

Local volatility surface derived from a Black vol surface.

`#include <ql/voltermstructure.hpp>`

`#include <ql/termstructure.hpp>`

Include dependency graph for localvolsurface.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::VolTermStructures**

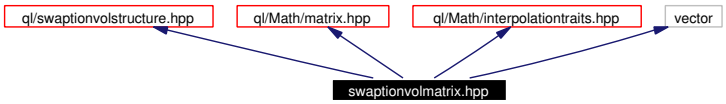
12.458 ql/Volatilities/swaptionvolmatrix.hpp File Reference

12.458.1 Detailed Description

Swaption at-the-money volatility matrix.

```
#include "ql/swaptionvolstructure.hpp"
#include "ql/Math/matrix.hpp"
#include "ql/Math/interpolationtraits.hpp"
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::Volatilities**

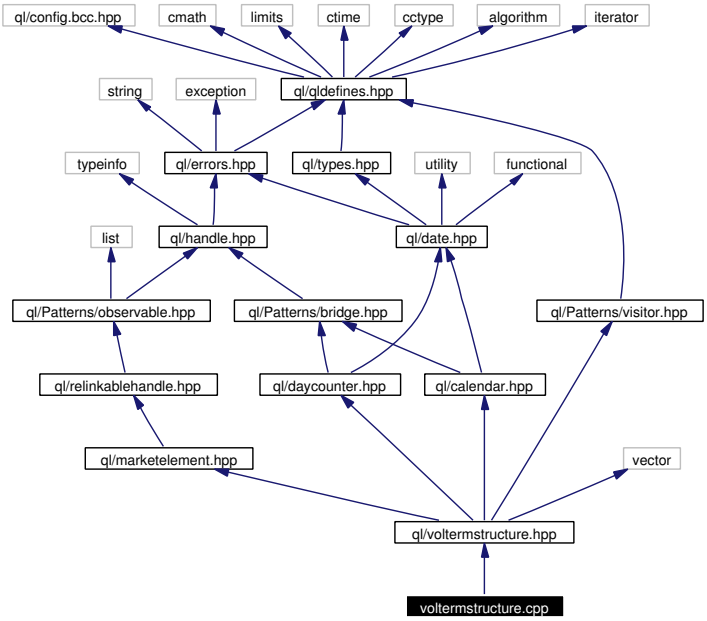
12.459 ql/voltermstructure.cpp File Reference

12.459.1 Detailed Description

Volatility term structures.

`#include <ql/voltermstructure.hpp>`

Include dependency graph for voltermstructure.cpp:



Namespaces

- namespace **QuantLib**

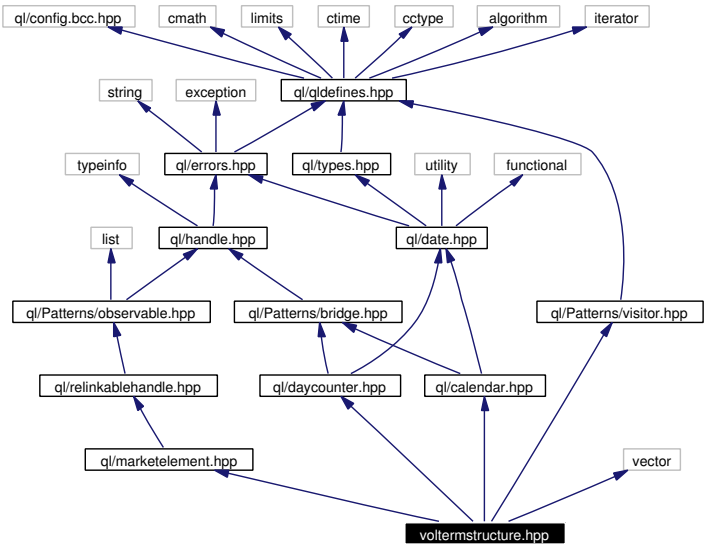
12.460 ql/voltermstructure.hpp File Reference

12.460.1 Detailed Description

Volatility term structures.

```
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/marketelement.hpp>
#include <ql/Patterns/visitor.hpp>
#include <vector>
```

Include dependency graph for voltermstructure.hpp:



Namespaces

- namespace **QuantLib::VolTermStructures**
- namespace **QuantLib**

Chapter 13

QuantLib Example Documentation

13.1 AmericanOption.cpp

This example calculates American options using different methods

```
#include <ql/quantlib.hpp>

using namespace QuantLib;
using namespace QuantLib::PricingEngines;
using namespace QuantLib::Math;
using namespace QuantLib::MonteCarlo;
using namespace QuantLib::RandomNumbers;
using namespace QuantLib::Pricers;
using namespace QuantLib::TermStructures;
using namespace QuantLib::VolTermStructures;


int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Put);
        double underlying = 36;
        double strike = 40;
        Spread dividendYield = 0.00;
        Rate riskFreeRate = 0.06;
        double volatility = 0.20;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Date exerciseDate(17, May, 1999);
        DayCounter rateDayCounter = DayCounters::Actual365();
        Time maturity = rateDayCounter.yearFraction(settlementDate,
                                                    exerciseDate);

        std::cout << "Time to maturity = "        << maturity
                  << std::endl;
        std::cout << "Underlying price = "        << underlying
                  << std::endl;
        std::cout << "Strike = "                  << strike
                  << std::endl;
        std::cout << "Risk-free interest rate = " << riskFreeRate
```

```
          << std::endl;
std::cout << "dividend yield = " << dividendYield
          << std::endl;
std::cout << "Volatility = "              << volatility
          << std::endl;
std::cout << std::endl;

std::string method;

double value, discrepancy, rightValue, relativeDiscrepancy;
rightValue = (type == Option::Put ? 4.4867 : 2.1737);

std::cout << std::endl ;

// write column headings
std::cout << "Method\t\tValue\tEstimatedError\tDiscrepancy"
    "\tRel. Discr." << std::endl;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

EuropeanExercise exercise(exerciseDate);
AmericanExercise amExercise(settlementDate, exerciseDate);
BermudanExercise berExercise(exDates);


RelinkableHandle<MarketElement> underlyingH(
    Handle<MarketElement>(new SimpleMarketElement(underlying)));

// bootstrap the yield/dividend/vol curves
RelinkableHandle<TermStructure> flatTermStructure(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        riskFreeRate, rateDayCounter)));
RelinkableHandle<TermStructure> flatDividendTS(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        dividendYield, rateDayCounter)));
RelinkableHandle<BlackVolTermStructure> flatVolTS(
    Handle<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility)));

std::vector<Date> dates(4);
dates[0] = settlementDate.plusMonths(1);
dates[1] = exerciseDate;
dates[2] = exerciseDate.plusMonths(6);
dates[3] = exerciseDate.plusMonths(12);
std::vector<double> strikes(4);
strikes[0] = underlying*0.9;
strikes[1] = underlying;
strikes[2] = underlying*1.1;
strikes[3] = underlying*1.2;

Matrix vols(4,4);
vols[0][0] = volatility*1.1; vols[0][1] = volatility;
    vols[0][2] = volatility*0.9; vols[0][3] = volatility*0.8;
vols[1][0] = volatility*1.1; vols[1][1] = volatility;
    vols[1][2] = volatility*0.9; vols[1][3] = volatility*0.8;
vols[2][0] = volatility*1.1; vols[2][1] = volatility;
    vols[2][2] = volatility*0.9; vols[2][3] = volatility*0.8;
vols[3][0] = volatility*1.1; vols[3][1] = volatility;
    vols[3][2] = volatility*0.9; vols[3][3] = volatility*0.8;
RelinkableHandle<BlackVolTermStructure> blackSurface(
    Handle<BlackVolTermStructure>(
        new BlackVarianceSurface(settlementDate, dates, strikes, vols)));
```

```
// European option
Instruments::VanillaOption euroOption(
    type,
    underlyingH,
    strike,
    flatDividendTS,
    flatTermStructure,
    exercise,
    flatVolTS,
    Handle<PricingEngine>(new AnalyticEuropeanEngine()));

// method: Black Scholes Engine
method = "european ";
value = euroOption.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;

// American option
Instruments::VanillaOption option(
    type,
    underlyingH,
    strike,
    flatDividendTS,
    flatTermStructure,
    amExercise,
    flatVolTS);

Size timeSteps = 800;

// Binomial Method (JR)
method = "Binomial (JR)";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine(
        BinomialVanillaEngine::JarrowRudd, timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;


// Binomial Method (CRR)
method = "Binomial (CRR)";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine(
        BinomialVanillaEngine::CoxRossRubinstein, timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;
```

```
        // Equal Probability Additive Binomial Tree (EQP)
        method = "Additive (EQP)";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::EQP, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // Equal Jumps Additive Binomial Tree (Trigeorgis)
        method = "Bin. Trigeorgis";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::Trigeorgis, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // Tian Binomial Tree (third moment matching)
        method = "Binomial Tian";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::Tian, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // Least Squares Monte Carlo: Longstaff Schwartz
        method = "LSMC (monomial)";
        Size mcSeed = 12345;
        Size nSamples = 5000;
        timeSteps = 100;
        option.setPricingEngine(Handle<PricingEngine>(
            new AmericanMCVanillaEngine(nSamples, timeSteps, mcSeed)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
            << DoubleFormatter::toString(value, 4) << "\t"
            << "N/A\t\t"
            << DoubleFormatter::toString(discrepancy, 6) << "\t"
            << DoubleFormatter::toString(relativeDiscrepancy, 6)
            << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
```

```
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

13.2 BermudanSwaption.cpp

This is an example of using the QuantLib short rate models.

```
#include <ql/quantlib.hpp>

using namespace QuantLib;
using namespace QuantLib::DayCounters;
using namespace QuantLib::Indexes;
using namespace QuantLib::Instruments;
using namespace QuantLib::Pricers;
using namespace QuantLib::ShortRateModels;
using namespace QuantLib::TermStructures;

using CalibrationHelpers::CapHelper;
using CalibrationHelpers::SwaptionHelper;


//Number of swaptions to be calibrated to...

Size numRows = 5;
Size numCols = 10;

unsigned int swaptionLengths[] = {1,2,3,4,5,7,10,15,20,25,30};
double swaptionVols[] = {
    23.92, 22.80, 19.8, 18.1, 16.0, 14.26, 13.56, 12.79, 12.3, 11.09,
    21.85, 21.50, 19.5, 17.2, 14.7, 13.23, 12.59, 12.29, 11.1, 10.30,
    19.46, 19.40, 17.9, 15.9, 13.9, 12.69, 12.15, 11.83, 10.8, 10.00,
    17.97, 17.80, 16.7, 14.9, 13.4, 12.28, 11.89, 11.48, 10.5,  9.80,
    16.29, 16.40, 15.1, 14.0, 12.9, 12.01, 11.46, 11.08, 10.4,  9.77,
    14.71, 14.90, 14.3, 13.2, 12.3, 11.49, 11.12, 10.70, 10.1,  9.57,
    12.93, 13.30, 12.8, 12.2, 11.6, 10.82, 10.47, 10.21,  9.8,  9.51,
    12.70, 12.10, 11.9, 11.2, 10.8, 10.40, 10.20, 10.00,  9.5,  9.00,
    12.30, 11.60, 11.6, 10.9, 10.5, 10.30, 10.00,  9.80,  9.3,  8.80,
    12.00, 11.40, 11.5, 10.8, 10.3, 10.00,  9.80,  9.60,  9.5,  9.10,
    11.50, 11.20, 11.3, 10.6, 10.2, 10.10,  9.70,  9.50,  9.4,  8.60};

void calibrateModel(const Handle<Model>& model,
                    CalibrationSet& calibs,
                    double lambda) {

    Optimization::Simplex om(lambda, 1e-9);
    om.setEndCriteria(Optimization::EndCriteria(10000, 1e-7));
    model->calibrate(calibs, om);

    #if defined(QL_PATCH_DARWIN)
    // to be investigated
    return;
    #endif

    //Output the implied Black volatilities
    Size i;
    for (i=0; i<numRows; i++) {
        std::cout << IntegerFormatter::toString(swaptionLengths[i],2) << "y|";
        for (Size j=0; j<numCols; j++) {
            Size k = i*numCols + j;
            double npv = calibs[k]->modelValue();
            double implied = calibs[k]->impliedVolatility(npv, 1e-4,
                1000, 0.05, 0.50)*100.0;
            std::cout << DoubleFormatter::toString(implied,1,4) << " (";
            k = i*10 + j;
            double diff = implied - swaptionVols[k];
            std::cout << DoubleFormatter::toString(diff,1,4)
                      << ")|";
        }
        std::cout << std::endl;
```

```
    }
}

int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        Date todaysDate(15, February, 2002);
        Calendar calendar = Calendars::TARGET();

        // Date settlementDate = calendar.advance(todaysDate,
        //                                        settlementDays, Days);
        Date settlementDate(19, February, 2002);

        //Instruments used to bootstrap the yield curve:
        std::vector<Handle<RateHelper> > instruments;

        //Deposit rates
        DayCounter depositDayCounter = Thirty360();
        int settlementDays = 2;

        Rate weekRates[3] = {3.295, 3.3, 3.3};
        Size i;
        for (i=0; i<3; i++) {
            RelinkableHandle<MarketElement> depositRate(
                Handle<MarketElement>(
                    new SimpleMarketElement(weekRates[i]*0.01)));
            Handle<RateHelper> depositHelper(new DepositRateHelper(
                depositRate, i+1, Weeks, settlementDays, calendar,
                ModifiedFollowing, depositDayCounter));
            instruments.push_back(depositHelper);
        }

        Rate depositRates[12] = {
            3.31, 3.32, 3.29, 3.35, 3.315, 3.33,
            3.395, 3.41, 3.41, 3.49, 3.54, 3.53};

        for (i=0; i<11; i++) {
            RelinkableHandle<MarketElement> depositRate(
                Handle<MarketElement>(
                    new SimpleMarketElement(depositRates[i]*0.01)));
            Handle<RateHelper> depositHelper(new DepositRateHelper(
                depositRate, i+1, Months, settlementDays, calendar,
                ModifiedFollowing, depositDayCounter));
            instruments.push_back(depositHelper);
        }

        //Swap rates
        Rate swapRates[13] = {
            3.6425, 4.0875, 4.38, 4.5815, 4.74325, 4.87375,
            4.9775, 5.07, 5.13, 5.1825, 5.36, 5.45125, 5.43875};
        int swapYears[13] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30};

        int swFixedLegFrequency = 1;
        bool swFixedLegIsAdjusted = false;
        DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
        int swFloatingLegFrequency = 2;

        for (i=0; i<13; i++) {
            Handle<MarketElement> swapRate(
                new SimpleMarketElement(swapRates[i]*0.01));
            Handle<RateHelper> swapHelper(new SwapRateHelper(
                RelinkableHandle<MarketElement>(swapRate),
                swapYears[i], Years, settlementDays,
                calendar, ModifiedFollowing,
                swFixedLegFrequency,
```

```
        swFixedLegIsAdjusted, swFixedLegDayCounter,
        swFloatingLegFrequency));
    instruments.push_back(swapHelper);
}


// bootstrapping the yield curve
Handle<PiecewiseFlatForward> myTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate, instruments,
                         depositDayCounter));

RelinkableHandle<TermStructure> rhTermStructure;
rhTermStructure.linkTo(myTermStructure);

//Define the ATM/OTM/ITM swaps
int fixedLegFrequency = 1;
bool fixedLegIsAdjusted = false;
RollingConvention roll = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
int floatingLegFrequency = 2;
bool payFixedRate = true;
int fixingDays = 2;
Rate dummyFixedRate = 0.03;
Handle<Xibor> indexSixMonths(new Euribor(6, Months, rhTermStructure));

Handle<SimpleSwap> swap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, dummyFixedRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Rate fixedATMRate = swap->fairRate();

Handle<SimpleSwap> atmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Handle<SimpleSwap> otmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate * 1.2,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Handle<SimpleSwap> itmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate * 0.8,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));

std::vector<Period> swaptionMaturities;
swaptionMaturities.push_back(Period(1, Months));
swaptionMaturities.push_back(Period(3, Months));
swaptionMaturities.push_back(Period(6, Months));
swaptionMaturities.push_back(Period(1, Years));
swaptionMaturities.push_back(Period(2, Years));
swaptionMaturities.push_back(Period(3, Years));
swaptionMaturities.push_back(Period(4, Years));
swaptionMaturities.push_back(Period(5, Years));
swaptionMaturities.push_back(Period(7, Years));
swaptionMaturities.push_back(Period(10, Years));

CalibrationSet swaptions;

//List of times that have to be included in the timegrid
std::list<Time> times;

for (i=0; i<numRows; i++) {
    for (unsigned int j=0; j<numCols; j++) {
```

```
        unsigned int k = i*10 + j;
        Handle<MarketElement> vol(
            new SimpleMarketElement(swaptionVols[k]*0.01));
        swaptions.push_back(Handle<CalibrationHelper>(
            new SwaptionHelper(swaptionMaturities[j],
                               Period(swaptionLengths[i], Years),
                               RelinkableHandle<MarketElement>(vol),
                               indexSixMonths,
                               rhTermStructure)));
        swaptions.back()->addTimesTo(times);
    }
}
const std::vector<Time> termTimes = myTermStructure->times();
for (i=0; i<termTimes.size(); i++)
    times.push_back(termTimes[i]);
// Building time-grid
TimeGrid grid(times.begin(), times.end(), 30);

Handle<Model> modelHW(new HullWhite(rhTermStructure));
Handle<Model> modelHW2(new HullWhite(rhTermStructure));
Handle<Model> modelBK(new BlackKarasinski(rhTermStructure));

std::cout << "Calibrating to swaptions" << std::endl;

std::cout << "Hull-White (analytic formulae):" << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new JamshidianSwaption(modelHW)));


calibrateModel(modelHW, swaptions, 0.05);
std::cout << "calibrated to "
          << modelHW->params()
          << std::endl
          << std::endl;

std::cout << "Hull-White (numerical calibration):" << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW2, grid)));


calibrateModel(modelHW2, swaptions, 0.05);
std::cout << "calibrated to "
          << modelHW2->params()
          << std::endl
          << std::endl;

std::cout << "Black-Karasinski: " << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelBK, grid)));
calibrateModel(modelBK, swaptions, 0.05);
std::cout << "calibrated to "
          << modelBK->params()
          << std::endl
          << std::endl;

std::cout << "Pricing an ATM bermudan swaption" << std::endl;

//Define the bermudan swaption
std::vector<Date> bermudanDates;
const std::vector<Handle<CashFlow> >& leg = swap->floatingLeg();
for (i=0; i<leg.size(); i++) {
    Handle<CashFlows::Coupon> coupon = leg[i];
    bermudanDates.push_back(coupon->accrualStartDate());
}

Instruments::Swaption bermudanSwaption(atmSwap,
    BermudanExercise(bermudanDates), rhTermStructure,
```

```
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

        //Do the pricing for each model
        bermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));
        std::cout << "HW:       " << bermudanSwaption.NPV() << std::endl;

        bermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
        std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;

        bermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
        std::cout << "BK:       " << bermudanSwaption.NPV() << std::endl;

        std::cout << "Pricing an OTM bermudan swaption" << std::endl;

        Instruments::Swaption otmBermudanSwaption(otmSwap,
            BermudanExercise(bermudanDates), rhTermStructure,
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

        //Do the pricing for each model
        otmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));
        std::cout << "HW:       " << otmBermudanSwaption.NPV() << std::endl;

        otmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
        std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;

        otmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
        std::cout << "BK:       " << otmBermudanSwaption.NPV() << std::endl;

        std::cout << "Pricing an ITM bermudan swaption" << std::endl;

        Instruments::Swaption itmBermudanSwaption(itmSwap,
            BermudanExercise(bermudanDates), rhTermStructure,
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

        //Do the pricing for each model
        itmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));
        std::cout << "HW:       " << itmBermudanSwaption.NPV() << std::endl;
        itmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
        std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
        itmBermudanSwaption.setPricingEngine(
            Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
        std::cout << "BK:       " << itmBermudanSwaption.NPV() << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

13.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```
/*  This example computes profit and loss of a discrete interval hedging
    strategy and compares with the results of Derman & Kamal's (Goldman Sachs
    Equity Derivatives Research) Research Note: "When You Cannot Hedge
    Continuously: The Corrections to Black-Scholes"
    (http://www.gs.com/qs/doc/when_you_cannot_hedge.pdf)

    Suppose an option hedger sells an European option and receives the
    Black-Scholes value as the options premium.
    Then he follows a Black-Scholes hedging strategy, rehedging at discrete,
    evenly spaced time intervals as the underlying stock changes. At
    expiration, the hedger delivers the option payoff to the option holder,
    and unwinds the hedge. We are interested in understanding the final
    profit or loss of this strategy.

    If the hedger had followed the exact Black-Scholes replication strategy,
    re-hedging continuously as the underlying stock evolved towards its final
    value at expiration, then, no matter what path the stock took, the final
    P&L would be exactly zero. When the replication strategy deviates from
    the exact Black-Scholes method, the final P&L may deviate from zero. This
    deviation is called the replication error. When the hedger rebalances at
    discrete rather than continuous intervals, the hedge is imperfect and the
    replication is inexact. The more often hedging occurs, the smaller the
    replication error.

    We examine the range of possibilities, computing the replication error.
*/

// the only header you need to use QuantLib
#include <ql/quantlib.hpp>


// introducing the players ....

// Rate and Time are just double, but having their own types allows for
// a stronger check at compile time
using QuantLib::Rate;
using QuantLib::Time;

// PlainVanillaPayoff is a helper class that calculates option payoff
using QuantLib::PlainVanillaPayoff;

// Option is a helper class that holds the enumeration {Call, Put, Straddle}
using QuantLib::Option;

// Handle is the QuantLib way to have reference-counted objects
using QuantLib::Handle;

// the classic Black Scholes analytic solution for European Option
using QuantLib::Pricers::EuropeanOption;

// class for statisticsal analysis
using QuantLib::Math::Statistics;

// single Path of a random variable
// It contains the list of variations
using QuantLib::MonteCarlo::Path;

// the pricer computes final portfolio's value for each random variable path
using QuantLib::MonteCarlo::PathPricer_old;

// the path generator
using QuantLib::MonteCarlo::GaussianPathGenerator_old;
```

```
// the Montecarlo pricing model for option on a single asset
using QuantLib::MonteCarlo::OneFactorMonteCarloOption_old;

// to format the output of doubles
using QuantLib::DoubleFormatter;

/* The ReplicationError class carries out Monte Carlo simulations to evaluate
   the outcome (the replication error) of the discrete hedging strategy over
   different, randomly generated scenarios of future stock price evolution.
*/
class ReplicationError
{
public:
    ReplicationError(Option::Type type,
                     Time maturity,
                     double strike,
                     double s0,
                     double sigma,
                     Rate r)
    : maturity_(maturity), payoff_(type, strike), s0_(s0),
      sigma_(sigma), r_(r) {

        // value of the option
        EuropeanOption option = EuropeanOption(type, s0_, strike, 0.0, r_,
            maturity_, sigma_);
        std::cout << "Option value: " << option.value() << std::endl;

        // store option's vega, since Derman and Kamal's formula needs it
        vega_ = option.vega();

        std::cout << std::endl;
        std::cout <<
            "        |        | P&L  \t|  P&L    | Derman&Kamal | P&L"
            "      \t| P&L" << std::endl;

        std::cout <<
            "samples | trades | Mean \t| Std Dev | Formula      |"
            " skewness \t| kurt." << std::endl;

        std::cout << "---------------------------------"
            "-----------------------------------------------" << std::endl;
    }

    // the actual replication error computation
    void compute(int nTimeSteps, int nSamples);
private:
    Time maturity_;
    PlainVanillaPayoff payoff_;
    double s0_;
    double sigma_;
    Rate r_;
    double vega_;
};

// The key for the MonteCarlo simulation is to have a PathPricer_old that
// implements a value(const Path& path) method.
// This method prices the portfolio for each Path of the random variable
class ReplicationPathPricer : public PathPricer_old<Path>
{
  public:
    // real constructor
    ReplicationPathPricer(Option::Type type,
                          double underlying,
                          double strike,
                          Rate r,
                          Time maturity,
```

```
                          double sigma)
    : PathPricer_old<Path>(1.0, false), type_(type), underlying_(underlying),
      strike_(strike), r_(r), maturity_(maturity), sigma_(sigma) {
        QL_REQUIRE(strike_ > 0.0,
            "ReplicationPathPricer: strike must be positive");
        QL_REQUIRE(underlying_ > 0.0,
            "ReplicationPathPricer: underlying must be positive");
        QL_REQUIRE(r_ >= 0.0,
            "ReplicationPathPricer: risk free rate (r) must"
            " be positive or zero");
        QL_REQUIRE(maturity_ > 0.0,
            "ReplicationPathPricer: maturity must be positive");
        QL_REQUIRE(sigma_ >= 0.0,
            "ReplicationPathPricer: volatility (sigma)"
            " must be positive or zero");

    }
    // The value() method encapsulates the pricing code
    double operator()(const Path& path) const;

  private:
    Option::Type type_;
    double underlying_, strike_;
    Rate r_;
    Time maturity_;
    double sigma_;
};


// Compute Replication Error as in the Derman and Kamal's research note
int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        Time maturity = 1./12.;   // 1 month
        double strike = 100;
        double underlying = 100;
        double volatility = 0.20; // 20%
        Rate riskFreeRate = 0.05; // 5%
        ReplicationError rp(Option::Call, maturity, strike, underlying,
                volatility, riskFreeRate);

        int scenarios = 50000;
        int hedgesNum;

        hedgesNum = 21;
        rp.compute(hedgesNum, scenarios);

        hedgesNum = 84;
        rp.compute(hedgesNum, scenarios);

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}


/* The actual computation of the Profit&Loss for each single path.

   In each scenario N rehedging trades spaced evenly in time over
   the life of the option are carried out, using the Black-Scholes
```

```
   hedge ratio.
*/
double ReplicationPathPricer::operator()(const Path& path) const
{

    // path is an instance of QuantLib::MonteCarlo::Path
    // It contains the list of variations.
    // It can be used as an array: it has a size() method
    int n = path.size();
    QL_REQUIRE(n>0,
        "ReplicationPathPricer: the path cannot be empty");

    // discrete hedging interval
    Time dt = maturity_/n;

    // For simplicity, we assume the stock pays no dividends.
    double stockDividendYield = 0.0;

    // let's start
    Time t = 0;

    // stock value at t=0
    double stock = underlying_;
    double stockLogGrowth = 0.0;

    // money account at t=0
    double money_account = 0.0;

    /************************/
    /*** the initial deal ***/
    /************************/
    // option fair price (Black-Scholes) at t=0
    EuropeanOption option = EuropeanOption(type_, stock, strike_,
        stockDividendYield, r_, maturity_, sigma_);
    // sell the option, cash in its premium
    money_account += option.value();
    // compute delta
    double delta = option.delta();
    // delta-hedge the option buying stock
    double stockAmount = delta;
    money_account -= stockAmount*stock;

    /*********************************/
    /*** hedging during option life ***/
    /*********************************/
    for(int step = 0; step < n-1; step++){

        // time flows
        t += dt;

        // accruing on the money account
        money_account *= QL_EXP( r_*dt );

        // stock growth:
        // path contains the list of Gaussian variations
        // and path[n] is the n-th variation
        stockLogGrowth += path[step];
        stock = underlying_*QL_EXP(stockLogGrowth);

        // recalculate option value at the current stock value,
        // and the current time to maturity
        EuropeanOption option = EuropeanOption(type_, stock, strike_,
            stockDividendYield, r_, maturity_-t, sigma_);

        // recalculate delta
        delta = option.delta();
```

```
        // re-hedging
        money_account -= (delta - stockAmount)*stock;
        stockAmount = delta;
    }

    /*************************/
    /*** option expiration ***/
    /*************************/
    // last accrual on my money account
    money_account *= QL_EXP( r_*dt );
    // last stock growth
    stockLogGrowth += path[n-1];
    stock = underlying_*QL_EXP(stockLogGrowth);

    // the hedger delivers the option payoff to the option holder
    double optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
    money_account -= optionPayoff;

    // and unwinds the hedge selling his stock position
    money_account += stockAmount*stock;

    // final Profit&Loss
    return money_account;
}


// The computation over nSamples paths of the P&L distribution
void ReplicationError::compute(int nTimeSteps, int nSamples)
{
    QL_REQUIRE(nTimeSteps>0,
        "ReplicationError::compute : the number of steps must be > 0");

    // hedging interval
    // double tau = maturity_ / nTimeSteps;

    /* Black-Scholes framework: the underlying stock price evolves
       lognormally with a fixed known volatility that stays constant
       throughout time.
    */
    double drift = r_ - 0.5*sigma_*sigma_;

    // Black Scholes equation rules the path generator:
    // at each step the log of the stock
    // will have drift and sigma^2 variance
    Handle<GaussianPathGenerator_old> myPathGenerator(
        new GaussianPathGenerator_old(drift, sigma_*sigma_,
            maturity_, nTimeSteps));

    // The replication strategy's Profit&Loss is computed for each path
    // of the stock. The path pricer knows how to price a path using its
    // value() method
    Handle<PathPricer_old<Path> > myPathPricer =
        Handle<PathPricer_old<Path> >(new
            ReplicationPathPricer(payoff_.optionType(), s0_,
                payoff_.strike(), r_, maturity_, sigma_));

    // a statistics accumulator for the path-dependant Profit&Loss values
    Statistics statisticsAccumulator;

    // The OneFactorMontecarloModel generates paths using myPathGenerator
    // each path is priced using myPathPricer
    // prices will be accumulated into statisticsAccumulator
    OneFactorMonteCarloOption_old MCSimulation(myPathGenerator,
        myPathPricer, statisticsAccumulator, false);

    // the model simulates nSamples paths
    MCSimulation.addSamples(nSamples);
```

```
    // the sampleAccumulator method of OneFactorMonteCarloOption_old
    // gives access to all the methods of statisticsAccumulator
    double PLMean  = MCSimulation.sampleAccumulator().mean();
    double PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
    double PLSkew  = MCSimulation.sampleAccumulator().skewness();
    double PLKurt  = MCSimulation.sampleAccumulator().kurtosis();

    // Derman and Kamal's formula
    double theorStD = QL_SQRT(M_PI/4/nTimeSteps)*vega_*sigma_;


    std::cout << nSamples << "\t| "
        << nTimeSteps << "\t | "
        << DoubleFormatter::toString(PLMean,   3) << " \t| "
        << DoubleFormatter::toString(PLStDev,  2) << " \t  | "
        << DoubleFormatter::toString(theorStD, 2) << " \t | "
        << DoubleFormatter::toString(PLSkew,   2) << " \t| "
        << DoubleFormatter::toString(PLKurt,   2) << std::endl;
}
```

13.4 EuropeanOption.cpp

This example calculates European options using different methods while testing call-put parity.

```
#include <ql/quantlib.hpp>

using namespace QuantLib;
using namespace QuantLib::Instruments;
using namespace QuantLib::PricingEngines;
using namespace QuantLib::Math;
using namespace QuantLib::MonteCarlo;
using namespace QuantLib::RandomNumbers;
using namespace QuantLib::Pricers;
using namespace QuantLib::TermStructures;
using namespace QuantLib::VolTermStructures;

// This will be included in the library after a bit of redesign
class WeightedPayoff {
    public:
        WeightedPayoff(Option::Type type,
               Time maturity,
               double strike,
               double s0,
               double sigma,
               Rate r,
               Rate q)
        : type_(type), maturity_(maturity),
        strike_(strike),
        s0_(s0),
        sigma_(sigma),r_(r), q_(q){}

        double operator()(double x) const {
           double nuT = (r_-q_-0.5*sigma_*sigma_)*maturity_;
           return QL_EXP(-r_*maturity_)
               *PlainVanillaPayoff(type_, strike_)(s0_*QL_EXP(x))
               *QL_EXP(-(x - nuT)*(x -nuT)/(2*sigma_*sigma_*maturity_))
               /QL_SQRT(2.0*M_PI*sigma_*sigma_*maturity_);
        }
private:
    Option::Type type_;
    Time maturity_;
    double strike_;
    double s0_;
    double sigma_;
    Rate r_,q_;
};


int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Call);
        double underlying = 7;
        double strike = 8;
        Spread dividendYield = 0.05;
        Rate riskFreeRate = 0.05;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Date exerciseDate(17, May, 1999);
        DayCounter rateDayCounter = DayCounters::Actual365();
```

```
Time maturity = rateDayCounter.yearFraction(settlementDate,
    exerciseDate);

double volatility = 0.10;
std::cout << "Time to maturity = "        << maturity
          << std::endl;
std::cout << "Underlying price = "        << underlying
          << std::endl;
std::cout << "Strike = "                  << strike
          << std::endl;
std::cout << "Risk-free interest rate = " << riskFreeRate
          << std::endl;
std::cout << "dividend yield = " << dividendYield
          << std::endl;
std::cout << "Volatility = "              << volatility
          << std::endl;
std::cout << std::endl;

std::string method;
double value, discrepancy, rightValue, relativeDiscrepancy;


std::cout << std::endl << std::endl ;

// write column headings
std::cout << "Method\t\tValue\tEstimatedError\tDiscrepancy"
    "\tRel. Discr." << std::endl;



// first method: Black Scholes analytic solution
method ="Black Scholes";
value = EuropeanOption(type, underlying, strike,
    dividendYield, riskFreeRate, maturity, volatility).value();
double estimatedError = 0.0;
discrepancy = 0.0;
relativeDiscrepancy = 0.0;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << DoubleFormatter::toString(estimatedError, 4) << "\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;


// store the Black Scholes value as the correct one
rightValue = value;




// second method: Call-Put parity
method ="Call-Put parity";
Option::Type reverseType =
    (type==Option::Call ? Option::Put : Option::Call);
double coefficient =
    (type==Option::Call ? 1.0 : -1.0);
value = EuropeanOption(reverseType, underlying, strike,
    dividendYield, riskFreeRate, maturity, volatility).value()
    + coefficient * (underlying*QL_EXP(-dividendYield*maturity)
    - strike*QL_EXP(- riskFreeRate*maturity));
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
```

```
          << discrepancy << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;


    // third method: Integral
    method ="Integral";
    using QuantLib::Math::SegmentIntegral;
    WeightedPayoff po(type, maturity, strike, underlying, volatility, riskFreeRate,
        dividendYield);
    SegmentIntegral integrator(5000);

    double nuT = (riskFreeRate - dividendYield
                  + 0.5*volatility*volatility)*maturity;
    double infinity = 10.0*volatility*QL_SQRT(maturity);

    value = integrator(po, nuT-infinity, nuT+infinity);
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
          << DoubleFormatter::toString(value, 4) << "\t"
          << "N/A\t\t"
          << DoubleFormatter::toString(discrepancy, 6) << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;




    // fourth method: Finite Differences
    method ="Finite Diff.";
    Size grid = 100;
    value = FdEuropean(type, underlying, strike,
        dividendYield, riskFreeRate, maturity, volatility, grid).value();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
          << DoubleFormatter::toString(value, 4) << "\t"
          << "N/A\t\t"
          << DoubleFormatter::toString(discrepancy, 6) << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;




    // fifth method: Monte Carlo (crude)
    method ="MC (crude)";
    Size mcSeed = 12345;
    bool antitheticVariance = false;
    McEuropean mcEur(type, underlying, strike, dividendYield,
        riskFreeRate, maturity, volatility, antitheticVariance, mcSeed);
    // let's require a tolerance of 0.002%
    value = mcEur.value(0.02);
    estimatedError = mcEur.errorEstimate();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
          << DoubleFormatter::toString(value, 4) << "\t"
          << DoubleFormatter::toString(estimatedError, 4) << "\t\t"
          << DoubleFormatter::toString(discrepancy, 6) << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;

    // sixth method: Monte Carlo with antithetic variance reduction
    method ="MC (antithetic)";
    // let's use the same number of samples as in the crude Monte Carlo
```

```
        Size nSamples = mcEur.sampleAccumulator().samples();
        antitheticVariance = true;
        McEuropean mcEur2(type, underlying, strike, dividendYield,
            riskFreeRate, maturity, volatility, antitheticVariance, mcSeed);
        value = mcEur2.valueWithSamples(nSamples);
        estimatedError = mcEur2.errorEstimate();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << DoubleFormatter::toString(estimatedError, 4) << "\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

/***********************************/

        // New option pricing framework
        std::cout << "\nNew Pricing engine framework" << std::endl;

        Date midlifeDate(19, November, 1998);
        std::vector<Date> exDates(2);
        exDates[0]=midlifeDate;
        exDates[1]=exerciseDate;

        EuropeanExercise exercise(exerciseDate);
        AmericanExercise amExercise(settlementDate, exerciseDate);
        BermudanExercise berExercise(exDates);


        RelinkableHandle<MarketElement> underlyingH(
            Handle<MarketElement>(new SimpleMarketElement(underlying)));

        // bootstrap the yield/dividend/vol curves
        RelinkableHandle<TermStructure> flatTermStructure(
            Handle<TermStructure>(
                new FlatForward(todaysDate, settlementDate,
                                riskFreeRate, rateDayCounter)));
        RelinkableHandle<TermStructure> flatDividendTS(
            Handle<TermStructure>(
                new FlatForward(todaysDate, settlementDate,
                                dividendYield, rateDayCounter)));
        RelinkableHandle<BlackVolTermStructure> flatVolTS(
            Handle<BlackVolTermStructure>(
                new BlackConstantVol(settlementDate, volatility)));

        std::vector<Date> dates(4);
        dates[0] = settlementDate.plusMonths(1);
        dates[1] = exerciseDate;
        dates[2] = exerciseDate.plusMonths(6);
        dates[3] = exerciseDate.plusMonths(12);
        std::vector<double> strikes(4);
        strikes[0] = underlying*0.9;
        strikes[1] = underlying;
        strikes[2] = underlying*1.1;
        strikes[3] = underlying*1.2;

        Matrix vols(4,4);
        vols[0][0] = volatility*1.1; vols[0][1] = volatility; vols[0][2] = volatility*0.9; vols[0][3] = volatility*0.8;
        vols[1][0] = volatility*1.1; vols[1][1] = volatility; vols[1][2] = volatility*0.9; vols[1][3] = volatility*0.8;
        vols[2][0] = volatility*1.1; vols[2][1] = volatility; vols[2][2] = volatility*0.9; vols[2][3] = volatility*0.8;
        vols[3][0] = volatility*1.1; vols[3][1] = volatility; vols[3][2] = volatility*0.9; vols[3][3] = volatility*0.8;

        RelinkableHandle<BlackVolTermStructure> blackSurface(
            Handle<BlackVolTermStructure>(
                new BlackVarianceSurface(settlementDate, dates, strikes, vols)));
```

```
        Instruments::VanillaOption option(
            type,
            underlyingH,
            strike,
            flatDividendTS,
            flatTermStructure,
            exercise,
            flatVolTS,
//            blackSurface,
            Handle<PricingEngine>(new AnalyticEuropeanEngine()));


        // method: Black Scholes Engine
        method = "Black Scholes";
        option.setPricingEngine(Handle<PricingEngine>(
            new AnalyticEuropeanEngine()));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;


        // method: Integral
        method = "Integral";
        option.setPricingEngine(Handle<PricingEngine>(
            new IntegralEngine()));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

/*
        // method: Integral
        method = "Binary Cash";
        option.setPricingEngine(Handle<PricingEngine>(
            new IntegralCashOrNothingEngine(1.0)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // method: Integral
        method = "Binary Asset";
        option.setPricingEngine(Handle<PricingEngine>(
            new IntegralAssetOrNothingEngine()));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
```

```
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

*/
        Size timeSteps=800;

        // Binomial Method (JR)
        method = "Binomial (JR)";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::JarrowRudd, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;


        // Binomial Method (CRR)
        method = "Binomial (CRR)";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::CoxRossRubinstein, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // Equal Probability Additive Binomial Tree (EQP)
        method = "Additive (EQP)";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::EQP, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;

        // Equal Jumps Additive Binomial Tree (Trigeorgis)
        method = "Bin. Trigeorgis";
        option.setPricingEngine(Handle<PricingEngine>(
            new BinomialVanillaEngine(
                BinomialVanillaEngine::Trigeorgis, timeSteps)));
        value = option.NPV();
        discrepancy = QL_FABS(value-rightValue);
        relativeDiscrepancy = discrepancy/rightValue;
        std::cout << method << "\t"
             << DoubleFormatter::toString(value, 4) << "\t"
             << "N/A\t\t"
             << DoubleFormatter::toString(discrepancy, 6) << "\t"
             << DoubleFormatter::toString(relativeDiscrepancy, 6)
             << std::endl;
```

```
// Tian Binomial Tree (third moment matching)
method = "Binomial Tian";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine(
        BinomialVanillaEngine::Tian, timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;

// Finite Differences Method: not implemented

/*method = "Finite Diff.";
option.setPricingEngine(Handle<PricingEngine>(
    new FDVanillaEngine()));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;*/


// Monte Carlo Method
timeSteps = 365;
TimeGrid timeGrid(maturity, timeSteps);
method = "MC (crude)";
Handle<PricingEngine> mcengine1(
    #if defined(QL_PATCH_MICROSOFT)
    /* the #else branch used to work--now Visual C++ needs this.
       We cannot go and see what's wrong because the other branch
       does work in debug mode...
    */
    new MCEuropeanEngine<PseudoRandom>(timeSteps, false, false,
                                       Null<int>(), 0.02,
                                       Null<int>(), mcSeed)
    #else
    MakeMCEuropeanEngine<PseudoRandom>().withStepsPerYear(timeSteps)
                                        .withTolerance(0.02)
                                        .withSeed(mcSeed)
    #endif
);
option.setPricingEngine(mcengine1);

value = option.NPV();
double errorEstimate = option.errorEstimate();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << DoubleFormatter::toString(errorEstimate, 4) << "\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;

method = "MC (Sobol)";

Handle<PricingEngine> mcengine2(
```

```
    #if defined(QL_PATCH_MICROSOFT)
    /* See above. */
    new MCEuropeanEngine<LowDiscrepancy>(timeSteps, false, false,
                                         nSamples, Null<double>(),
                                         Null<int>())
    #else
    MakeMCEuropeanEngine<LowDiscrepancy>().withStepsPerYear(timeSteps)
                                          .withSamples(nSamples)
    #endif
);
option.setPricingEngine(mcengine2);

value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
     << DoubleFormatter::toString(value, 4) << "\t"
     << "N/A\t\t"
     << DoubleFormatter::toString(discrepancy, 6) << "\t"
     << DoubleFormatter::toString(relativeDiscrepancy, 6)
     << std::endl;



// quanto and all that

Handle<AnalyticEuropeanEngine> baseEngine(new
    AnalyticEuropeanEngine);

Handle<QuantoEngine<VanillaOption::arguments,
                    VanillaOption::results> >
    quantoEngine(new QuantoEngine<VanillaOption::arguments,
                                  VanillaOption::results>(baseEngine));

double correlation = 0.0;
Instruments::QuantoVanillaOption quantoOption(
    type,
    underlyingH,
    strike,
    flatDividendTS,
    flatTermStructure,
    exercise,
    flatVolTS,
    quantoEngine,
    flatTermStructure,
    flatVolTS,
    RelinkableHandle<MarketElement>(
        Handle<MarketElement>(new SimpleMarketElement(correlation))));

value = quantoOption.NPV();
double delta = quantoOption.delta();
double gamma = quantoOption.gamma();
double theta = quantoOption.theta();
double vega = quantoOption.vega();
double rho = quantoOption.rho();
double divRho = quantoOption.dividendRho();
double qvega = quantoOption.qvega();
double qrho = quantoOption.qrho();
double qlambda = quantoOption.qlambda();
std::cout << std::endl << std::endl << "quanto: "
     << DoubleFormatter::toString(value, 4)
     << std::endl;
std::cout << "quanto delta: "
     << DoubleFormatter::toString(delta, 4)
     << std::endl;
std::cout << "quanto gamma: "
     << DoubleFormatter::toString(gamma, 4)
```

```
     << std::endl;
std::cout << "quanto theta: "
     << DoubleFormatter::toString(theta, 4)
     << std::endl;
std::cout << "quanto vega: "
     << DoubleFormatter::toString(vega, 4)
     << std::endl;
std::cout << "quanto rho: "
     << DoubleFormatter::toString(rho, 4)
     << std::endl;
std::cout << "quanto divRho: "
     << DoubleFormatter::toString(divRho, 4)
     << std::endl;
std::cout << "quanto qvega: "
     << DoubleFormatter::toString(qvega, 4)
     << std::endl;
std::cout << "quanto qrho: "
     << DoubleFormatter::toString(qrho, 4)
     << std::endl;
std::cout << "quanto qlambda: "
     << DoubleFormatter::toString(qlambda, 4)
     << std::endl;


Handle<ForwardEngine<VanillaOption::arguments,
                     VanillaOption::results> >
    forwardEngine(new ForwardEngine<VanillaOption::arguments,
                                    VanillaOption::results>(baseEngine));

Instruments::ForwardVanillaOption forwardOption(
    type,
    underlyingH,
    flatDividendTS,
    flatTermStructure,
    exercise,
    flatVolTS,
    forwardEngine,
    1.1, // moneyness
    settlementDate.plusMonths(1) // reset Date
    );

value   = forwardOption.NPV();
delta   = forwardOption.delta();
gamma   = forwardOption.gamma();
theta   = forwardOption.theta();
vega    = forwardOption.vega();
rho     = forwardOption.rho();
divRho  = forwardOption.dividendRho();
std::cout << std::endl << std::endl << "forward: "
     << DoubleFormatter::toString(value, 4)
     << std::endl;
std::cout << "forward delta: "
     << DoubleFormatter::toString(delta, 4)
     << std::endl;
std::cout << "forward gamma: "
     << DoubleFormatter::toString(gamma, 4)
     << std::endl;
std::cout << "forward theta: "
     << DoubleFormatter::toString(theta, 4)
     << std::endl;
std::cout << "forward vega: "
     << DoubleFormatter::toString(vega, 4)
     << std::endl;
std::cout << "forward rho: "
     << DoubleFormatter::toString(rho, 4)
     << std::endl;
std::cout << "forward divRho: "
```

```
     << DoubleFormatter::toString(divRho, 4)
     << std::endl;


Handle<ForwardPerformanceEngine<VanillaOption::arguments,
                                VanillaOption::results> >
    forwardPerformanceEngine(
        new ForwardPerformanceEngine<VanillaOption::arguments,
                                     VanillaOption::results>(
            baseEngine));

forwardOption.setPricingEngine(forwardPerformanceEngine);

value   = forwardOption.NPV();
delta   = forwardOption.delta();
gamma   = forwardOption.gamma();
theta   = forwardOption.theta();
vega    = forwardOption.vega();
rho     = forwardOption.rho();
divRho  = forwardOption.dividendRho();
std::cout << std::endl << std::endl << "forward performance: "
     << DoubleFormatter::toString(value, 4)
     << std::endl;
std::cout << "forward performance delta: "
     << DoubleFormatter::toString(delta, 4)
     << std::endl;
std::cout << "forward performance gamma: "
     << DoubleFormatter::toString(gamma, 4)
     << std::endl;
std::cout << "forward performance theta: "
     << DoubleFormatter::toString(theta, 4)
     << std::endl;
std::cout << "forward performance vega: "
     << DoubleFormatter::toString(vega, 4)
     << std::endl;
std::cout << "forward performance rho: "
     << DoubleFormatter::toString(rho, 4)
     << std::endl;
std::cout << "forward performance divRho: "
     << DoubleFormatter::toString(divRho, 4)
     << std::endl;



Handle<QuantoEngine<ForwardVanillaOption::arguments,
                    ForwardVanillaOption::results> >
    quantoForwardEngine(
       new QuantoEngine<ForwardVanillaOption::arguments,
                        ForwardVanillaOption::results>(forwardEngine));

Instruments::QuantoForwardVanillaOption quantoForwardOption(
    type,
    underlyingH,
    strike,
    flatDividendTS,
    flatTermStructure,
    exercise,
    flatVolTS,
    quantoForwardEngine,
    flatTermStructure,
    flatVolTS,
    RelinkableHandle<MarketElement>(
        Handle<MarketElement>(new SimpleMarketElement(correlation))),
    1.1, // moneyness
    settlementDate.plusMonths(1) // reset Date
);
```

```
value   = quantoForwardOption.NPV();
delta   = quantoForwardOption.delta();
gamma   = quantoForwardOption.gamma();
theta   = quantoForwardOption.theta();
vega    = quantoForwardOption.vega();
rho     = quantoForwardOption.rho();
divRho  = quantoForwardOption.dividendRho();
qvega   = quantoForwardOption.qvega();
qrho    = quantoForwardOption.qrho();
qlambda = quantoForwardOption.qlambda();

std::cout << std::endl << std::endl << "quanto forward: "
     << DoubleFormatter::toString(value, 4)
     << std::endl;
std::cout << "quanto forward delta: "
     << DoubleFormatter::toString(delta, 4)
     << std::endl;
std::cout << "quanto forward gamma: "
     << DoubleFormatter::toString(gamma, 4)
     << std::endl;
std::cout << "quanto forward theta: "
     << DoubleFormatter::toString(theta, 4)
     << std::endl;
std::cout << "quanto forward vega: "
     << DoubleFormatter::toString(vega, 4)
     << std::endl;
std::cout << "quanto forward rho: "
     << DoubleFormatter::toString(rho, 4)
     << std::endl;
std::cout << "quanto forward divRho: "
     << DoubleFormatter::toString(divRho, 4)
     << std::endl;
std::cout << "quanto forward qvega: "
     << DoubleFormatter::toString(qvega, 4)
     << std::endl;
std::cout << "quanto forward qrho: "
     << DoubleFormatter::toString(qrho, 4)
     << std::endl;
std::cout << "quanto forward qlambda: "
     << DoubleFormatter::toString(qlambda, 4)
     << std::endl;


Handle<QuantoEngine<ForwardVanillaOption::arguments,
                    ForwardVanillaOption::results> >
    quantoForwardPerformanceEngine(
       new QuantoEngine<ForwardVanillaOption::arguments,
                        ForwardVanillaOption::results>(
           forwardPerformanceEngine));
quantoForwardOption.setPricingEngine(quantoForwardPerformanceEngine);

value   = quantoForwardOption.NPV();
delta   = quantoForwardOption.delta();
gamma   = quantoForwardOption.gamma();
theta   = quantoForwardOption.theta();
vega    = quantoForwardOption.vega();
rho     = quantoForwardOption.rho();
divRho  = quantoForwardOption.dividendRho();
qvega   = quantoForwardOption.qvega();
qrho    = quantoForwardOption.qrho();
qlambda = quantoForwardOption.qlambda();

std::cout << std::endl << std::endl << "quanto forward performance: "
     << DoubleFormatter::toString(value, 4)
     << std::endl;
std::cout << "quanto forward performance delta: "
     << DoubleFormatter::toString(delta, 4)
```

```
            << std::endl;
        std::cout << "quanto forward performance gamma: "
            << DoubleFormatter::toString(gamma, 4)
            << std::endl;
        std::cout << "quanto forward performance theta: "
            << DoubleFormatter::toString(theta, 4)
            << std::endl;
        std::cout << "quanto forward performance vega: "
            << DoubleFormatter::toString(vega, 4)
            << std::endl;
        std::cout << "quanto forward performance rho: "
            << DoubleFormatter::toString(rho, 4)
            << std::endl;
        std::cout << "quanto forward performance divRho: "
            << DoubleFormatter::toString(divRho, 4)
            << std::endl;
        std::cout << "quanto forward performance qvega: "
            << DoubleFormatter::toString(qvega, 4)
            << std::endl;
        std::cout << "quanto forward performance qrho: "
            << DoubleFormatter::toString(qrho, 4)
            << std::endl;
        std::cout << "quanto forward performance qlambda: "
            << DoubleFormatter::toString(qlambda, 4)
            << std::endl;

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

13.5 history_iterators.cpp

This code exemplifies how to use History iterators to perform gaussianstatistics analyses on historical data.

```
// initialize a History
History h(...);

// print out the mean value and its standard deviation.

GaussianStatistics s;
s.addSequence(h.vdbegin(),h.vdend());
cout << "Historical mean: " << s.mean() << endl;
cout << "Std. deviation:  " << s.standardDeviation() << endl;

// Another possibility: print out the maximum value.

History::const_valid_iterator max = h.vbegin(), i=max, end = h.vend();
for (i++; i!=end; i++)
    if (i->value() > max->value())
        max = i;
cout << "Maximum value: " << max->value()
     << " assumed " << DateFormatter::toString(max->date()) << endl;

// or the minimum, this time the STL way:

bool lessthan(const History::Entry& i, const History::Entry& j) {
    return i.value() < j.value();
}

History::const_valid_iterator min =
    std::min_element(h.vbegin(),h.vend(),lessthan);
cout << "Minimum value: " << min->value()
     << " assumed " << DateFormatter::toString(min->date()) << endl;
```

13.6 swapvaluation.cpp

This is an example of using the QuantLib Term Structure for pricing a simple swap.

```
/*  This example shows how to set up a Term Structure and then price a simple
    swap.
*/

// the only header you need to use QuantLib
#include <ql/quantlib.hpp>


using namespace QuantLib;
using Calendars::TARGET;
using DayCounters::ActualActual;
using DayCounters::Actual360;
using DayCounters::Thirty360;
using Indexes::Xibor;
using Indexes::Euribor;
using Instruments::SimpleSwap;
using TermStructures::PiecewiseFlatForward;
using TermStructures::FlatForward;
using TermStructures::RateHelper;
using TermStructures::DepositRateHelper;
using TermStructures::FraRateHelper;
using TermStructures::FuturesRateHelper;
using TermStructures::SwapRateHelper;


int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        /*********************
         ***  MARKET DATA  ***
         *********************/

        Calendar calendar = TARGET();
        Date todaysDate(6, November, 2001);
        Date settlementDate(8, November, 2001);

        // deposits
        double d1wQuote=0.0382;
        double d1mQuote=0.0372;
        double d3mQuote=0.0363;
        double d6mQuote=0.0353;
        double d9mQuote=0.0348;
        double d1yQuote=0.0345;
        // FRAs
        double fra3x6Quote=0.037125;
        double fra6x9Quote=0.037125;
        double fra6x12Quote=0.037125;
        // futures
        double fut1Quote=96.2875;
        double fut2Quote=96.7875;
        double fut3Quote=96.9875;
        double fut4Quote=96.6875;
        double fut5Quote=96.4875;
        double fut6Quote=96.3875;
        double fut7Quote=96.2875;
        double fut8Quote=96.0875;
        // swaps
        double s2yQuote=0.037125;
        double s3yQuote=0.0398;
        double s5yQuote=0.0443;
```

```
double s10yQuote=0.05165;
double s15yQuote=0.055175;



/*********************
 ***  RATE HELPERS ***
 *********************/

// RateHelpers are built from the above quotes together with other
// instrument dependant infos.

// setup deposits
DayCounter depositDayCounter = Actual360();
int settlementDays = 2;

Handle<RateHelper> d1w(new DepositRateHelper(
    d1wQuote, 1, Weeks, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d1m(new DepositRateHelper(
    d1mQuote, 1, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d3m(new DepositRateHelper(
    d3mQuote, 3, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d6m(new DepositRateHelper(
    d6mQuote, 6, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d9m(new DepositRateHelper(
    d9mQuote, 9, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d1y(new DepositRateHelper(
    d1yQuote, 1, Years, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));

// setup swaps
int swFixedLegFrequency = 1;
bool swFixedLegIsAdjusted = false;
DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
int swFloatingLegFrequency = 2;

Handle<RateHelper> s2y(new SwapRateHelper(
    s2yQuote, 2, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s3y(new SwapRateHelper(
    s3yQuote, 3, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s5y(new SwapRateHelper(
    s5yQuote, 5, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s10y(new SwapRateHelper(
    s10yQuote, 10, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s15y(new SwapRateHelper(
    s15yQuote, 15, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
```

```
/*********************
 **  CURVE BUILDING **
 *********************/

// Any DayCounter would be fine.
// ActualActual::ISDA ensures that 30 years is 30.0
DayCounter termStructureDayCounter =
    ActualActual(ActualActual::ISDA);

// A depo-swap curve
std::vector<Handle<RateHelper> > depoSwapInstruments;
depoSwapInstruments.push_back(d1w);
depoSwapInstruments.push_back(d1m);
depoSwapInstruments.push_back(d3m);
depoSwapInstruments.push_back(d6m);
depoSwapInstruments.push_back(d9m);
depoSwapInstruments.push_back(d1y);
depoSwapInstruments.push_back(s2y);
depoSwapInstruments.push_back(s3y);
depoSwapInstruments.push_back(s5y);
depoSwapInstruments.push_back(s10y);
depoSwapInstruments.push_back(s15y);
Handle<TermStructure> depoSwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
    depoSwapInstruments, termStructureDayCounter));


 /*********************
 * SWAPS TO BE PRICED *
 *********************/

// Term structures that will be used for pricing:
// the one used for discounting cash flows
RelinkableHandle<TermStructure> discountingTermStructure;
// the one used for forward rate forecasting
RelinkableHandle<TermStructure> forecastingTermStructure;


// constant nominal 1,000,000 Euro
double nominal = 1000000.0;
// fixed leg
int fixedLegFrequency = 1; // annual
bool fixedLegIsAdjusted = false;
RollingConvention roll = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
int fixingDays = 2;
Rate fixedRate = 0.04;

// floating leg
int floatingLegFrequency = 2;
Handle<Xibor> euriborIndex(new Euribor(6, Months,
    forecastingTermStructure)); // using the forecasting curve
Spread spread = 0.0;

int lenghtInYears = 5;
bool payFixedRate = true;
SimpleSwap spot5YearSwap(payFixedRate, settlementDate, lenghtInYears,
    Years, calendar, roll, nominal, fixedLegFrequency, fixedRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    euriborIndex, fixingDays, spread,
    discountingTermStructure); // using the discounting curve
SimpleSwap oneYearForward5YearSwap(payFixedRate,
    calendar.advance(settlementDate, 1, Years, ModifiedFollowing),
    lenghtInYears, Years,
    calendar, roll, nominal, fixedLegFrequency, fixedRate,
```

```
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    euriborIndex, fixingDays, spread,
    discountingTermStructure); // using the discounting curve



 /***************
 * SWAP PRICING *
 ****************/

std::cout <<  "*** 5Y swap at 4.43%" << std::endl;


// let's price in term of NPV, fixed rate, and spread
double NPV;
Rate fairFixedRate;
Spread fairFloatingSpread;

// Of course, you're not forced to really use different curves
forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);
std::cout << "*** using Depo-Swap term structure:" << std::endl;

NPV = spot5YearSwap.NPV();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:               "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = spot5YearSwap.fairSpread();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:          "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = spot5YearSwap.fairRate();
std::cout << "5Y fair fixed rate:             "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;
// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairFixedRate-s5yQuote)<1e-8,
    "5 years swap mispriced!");

// now let's price the 1Y forward 5Y swap
NPV = oneYearForward5YearSwap.NPV();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:            "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = oneYearForward5YearSwap.fairSpread();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:       "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = oneYearForward5YearSwap.fairRate();
std::cout << "1Yx5Y fair fixed rate:          "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;




// now, let's get serious
```

```
/*********************
 ***  MARKET DATA  ***
 *********************/

// market elements are containers for quotes.
// SimpleMarketElement stores a value which can be manually changed;
// other MarketElement subclasses could read the value from a
// database or some kind of data feed.

// deposits
Handle<MarketElement> d1wRate(new SimpleMarketElement(d1wQuote));
Handle<MarketElement> d1mRate(new SimpleMarketElement(d1mQuote));
Handle<MarketElement> d3mRate(new SimpleMarketElement(d3mQuote));
Handle<MarketElement> d6mRate(new SimpleMarketElement(d6mQuote));
Handle<MarketElement> d9mRate(new SimpleMarketElement(d9mQuote));
Handle<MarketElement> d1yRate(new SimpleMarketElement(d1yQuote));
// FRAs
Handle<MarketElement> fra3x6Rate(new SimpleMarketElement(fra3x6Quote));
Handle<MarketElement> fra6x9Rate(new SimpleMarketElement(fra6x9Quote));
Handle<MarketElement> fra6x12Rate(new SimpleMarketElement(fra6x12Quote));
// futures
Handle<MarketElement> fut1Price(new SimpleMarketElement(fut1Quote));
Handle<MarketElement> fut2Price(new SimpleMarketElement(fut2Quote));
Handle<MarketElement> fut3Price(new SimpleMarketElement(fut3Quote));
Handle<MarketElement> fut4Price(new SimpleMarketElement(fut4Quote));
Handle<MarketElement> fut5Price(new SimpleMarketElement(fut5Quote));
Handle<MarketElement> fut6Price(new SimpleMarketElement(fut6Quote));
Handle<MarketElement> fut7Price(new SimpleMarketElement(fut7Quote));
Handle<MarketElement> fut8Price(new SimpleMarketElement(fut8Quote));
// swaps
Handle<MarketElement> s2yRate(new SimpleMarketElement(s2yQuote));
Handle<MarketElement> s3yRate(new SimpleMarketElement(s3yQuote));
Handle<MarketElement> s5yRate(new SimpleMarketElement(s5yQuote));
Handle<MarketElement> s10yRate(new SimpleMarketElement(s10yQuote));
Handle<MarketElement> s15yRate(new SimpleMarketElement(s15yQuote));




/*********************
 ***  RATE HELPERS ***
 *********************/

// RateHelpers are built from the above quotes together with other
// instrument dependant infos.
// This time quotes are passed in relinkable
// handles which could be relinked to some other data source later.

// setup deposits
d1w =Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d1wRate),
    1, Weeks, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
d1m=Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d1mRate),
    1, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
d3m=Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d3mRate),
    3, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
d6m=Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d6mRate),
    6, Months, settlementDays,
```

```
    calendar, ModifiedFollowing, depositDayCounter));
d9m=Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d9mRate),
    9, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
d1y=Handle<RateHelper>(new DepositRateHelper(
    RelinkableHandle<MarketElement>(d1yRate),
    1, Years, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));

// setup swaps
s2y=Handle<RateHelper>(new SwapRateHelper(
    RelinkableHandle<MarketElement>(s2yRate),
    2, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
s3y=Handle<RateHelper>(new SwapRateHelper(
    RelinkableHandle<MarketElement>(s3yRate),
    3, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
s5y=Handle<RateHelper>(new SwapRateHelper(
    RelinkableHandle<MarketElement>(s5yRate),
    5, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
s10y=Handle<RateHelper>(new SwapRateHelper(
    RelinkableHandle<MarketElement>(s10yRate),
    10, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
s15y=Handle<RateHelper>(new SwapRateHelper(
    RelinkableHandle<MarketElement>(s15yRate),
    15, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));


// let's add FRA and futures

// setup FRAs
Handle<RateHelper> fra3x6(new FraRateHelper(
    RelinkableHandle<MarketElement>(fra3x6Rate),
    3, 6, settlementDays, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fra6x9(new FraRateHelper(
    RelinkableHandle<MarketElement>(fra6x9Rate),
    6, 9, settlementDays, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fra6x12(new FraRateHelper(
    RelinkableHandle<MarketElement>(fra6x12Rate),
    6, 12, settlementDays, calendar, ModifiedFollowing,
    depositDayCounter));

// setup futures
int futMonths = 3;
Handle<RateHelper> fut1(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(19, December, 2001),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut2(new FuturesRateHelper(
```

```
    RelinkableHandle<MarketElement>(fut1Price),
    Date(20, March, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut3(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(19, June, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut4(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(18, September, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut5(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(18, December, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut6(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(19, March, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut7(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(18, June, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut8(new FuturesRateHelper(
    RelinkableHandle<MarketElement>(fut1Price),
    Date(17, September, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));




/*********************
 **  CURVE BUILDING **
 *********************/

// A depo-futures-swap curve
std::vector<Handle<RateHelper> > depoFutSwapInstruments;
depoFutSwapInstruments.push_back(d1w);
depoFutSwapInstruments.push_back(d1m);
depoFutSwapInstruments.push_back(fut1);
depoFutSwapInstruments.push_back(fut2);
depoFutSwapInstruments.push_back(fut3);
depoFutSwapInstruments.push_back(fut4);
depoFutSwapInstruments.push_back(fut5);
depoFutSwapInstruments.push_back(fut6);
depoFutSwapInstruments.push_back(fut7);
depoFutSwapInstruments.push_back(fut8);
depoFutSwapInstruments.push_back(s3y);
depoFutSwapInstruments.push_back(s5y);
depoFutSwapInstruments.push_back(s10y);
depoFutSwapInstruments.push_back(s15y);
Handle<TermStructure> depoFutSwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
    depoFutSwapInstruments, termStructureDayCounter));



// A depo-FRA-swap curve
```

```
std::vector<Handle<RateHelper> > depoFRASwapInstruments;
depoFRASwapInstruments.push_back(d1w);
depoFRASwapInstruments.push_back(d1m);
depoFRASwapInstruments.push_back(d3m);
depoFRASwapInstruments.push_back(fra3x6);
depoFRASwapInstruments.push_back(fra6x9);
depoFRASwapInstruments.push_back(fra6x12);
depoFRASwapInstruments.push_back(s2y);
depoFRASwapInstruments.push_back(s3y);
depoFRASwapInstruments.push_back(s5y);
depoFRASwapInstruments.push_back(s10y);
depoFRASwapInstruments.push_back(s15y);
Handle<TermStructure> depoFRASwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
    depoFRASwapInstruments, termStructureDayCounter));




 /***************
 * SWAP PRICING *
 ****************/


// switch the curve used by the swaps to be priced
forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);
std::cout << "*** using Depo-Fut-Swap term structure:" << std::endl;

NPV = spot5YearSwap.NPV();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:                "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = spot5YearSwap.fairSpread();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:          "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = spot5YearSwap.fairRate();
std::cout << "5Y fair fixed rate:             "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;
// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairFixedRate-s5yQuote)<1e-8,
    "5 years swap mispriced!");

// now let's price the 1Y forward 5Y swap
NPV = oneYearForward5YearSwap.NPV();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:            "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = oneYearForward5YearSwap.fairSpread();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:       "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = oneYearForward5YearSwap.fairRate();
std::cout << "1Yx5Y fair fixed rate:          "
    << RateFormatter::toString(fairFixedRate,4)
```

```
    << std::endl;


// switch the curve used by the swaps to be priced
forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);
std::cout << "*** using Depo-FRA-Swap term structure:" << std::endl;

NPV = spot5YearSwap.NPV();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:               "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = spot5YearSwap.fairSpread();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:          "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = spot5YearSwap.fairRate();
std::cout << "5Y fair fixed rate:             "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;
// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairFixedRate-s5yRate->value())<1e-8,
    "5 years swap mispriced!");

// now let's price the 1Y forward 5Y swap
NPV = oneYearForward5YearSwap.NPV();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:            "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = oneYearForward5YearSwap.fairSpread();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:       "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = oneYearForward5YearSwap.fairRate();
std::cout << "1Yx5Y fair fixed rate:          "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;


// now let's say that the 5-years swap rate goes up to 4.60%.
// A smarter market element--say, connected to a data source-- would
// notice the change itself. Since we're using SimpleMarketElements,
// we'll have to change the value manually--which forces us to
// downcast the handle and use the SimpleMarketElement
// interface. In any case, the point here is that a change in the
// value contained in the MarketElement triggers a new bootstrapping
// of the curve and a repricing of the swap.

Handle<SimpleMarketElement> fiveYearsRate = s5yRate;
fiveYearsRate->setValue(0.0460);
std::cout << std::endl <<  "*** 5Y swap goes up to 4.60%" << std::endl;


// now get the updated results
forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);
std::cout << "*** using Depo-Fut-Swap term structure:" << std::endl;

NPV = spot5YearSwap.NPV();
```

```
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:               "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = spot5YearSwap.fairSpread();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:          "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = spot5YearSwap.fairRate();
std::cout << "5Y fair fixed rate:             "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;
// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairFixedRate-s5yRate->value())<1e-8,
    "5 years swap mispriced!");

NPV = oneYearForward5YearSwap.NPV();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:            "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = oneYearForward5YearSwap.fairSpread();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:       "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = oneYearForward5YearSwap.fairRate();
std::cout << "1Yx5Y fair fixed rate:          "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;


forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);
std::cout << "*** using Depo-FRA-Swap term structure:" << std::endl;

NPV = spot5YearSwap.NPV();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:               "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
fairFloatingSpread = spot5YearSwap.fairSpread();
std::cout << "5Y "
    << RateFormatter::toString(fixedRate,2)
    << " fair spread:          "
    << RateFormatter::toString(fairFloatingSpread,4)
    << std::endl;
fairFixedRate = spot5YearSwap.fairRate();
std::cout << "5Y fair fixed rate:             "
    << RateFormatter::toString(fairFixedRate,4)
    << std::endl;
// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairFixedRate-s5yRate->value())<1e-8,
    "5 years swap mispriced!");

NPV = oneYearForward5YearSwap.NPV();
std::cout << "1Yx5Y "
    << RateFormatter::toString(fixedRate,2)
    << " NPV:            "
    << DoubleFormatter::toString(NPV,2)
    << std::endl;
```

```
        fairFloatingSpread = oneYearForward5YearSwap.fairSpread();
        std::cout << "1Yx5Y "
            << RateFormatter::toString(fixedRate,2)
            << " fair spread:       "
            << RateFormatter::toString(fairFloatingSpread,4)
            << std::endl;
        fairFixedRate = oneYearForward5YearSwap.fairRate();
        std::cout << "1Yx5Y fair fixed rate:          "
            << RateFormatter::toString(fairFixedRate,4)
            << std::endl;


        return 0;

    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

Chapter 14

Unstable features

Class AmericanMCVanillaEngine(p. 163) This engine is known not to work for deeply out-of-the-money options. More problems might surface.

Class CoxIngersollRoss(p. 278) This class was not tested enough to guarantee its functionality.

Class ExtendedCoxIngersollRoss(p. 336) This class was not tested enough to guarantee its functionality.

Class G2(p. 371) This class was not tested enough to guarantee its functionality.

Member sensitivity(p. 622)**(int basis=2) const** This method was not tested enough to guarantee its functionality.

Chapter 15

Deprecated List

Class BarrierOption(p. 184) use Instruments::BarrierOption instead.

Class BinaryOption(p. 199) use Instruments::BarrierOption instead.

Member BoxMullerGaussianRng(p. 227)**(long seed=0)** initialize with a random number generator instead.

Member CLGaussianRng(p. 256)**(long seed=0)** initialize with a random number generator instead.

Class CliquetOptionPathPricer_old(p. 262) use CliquetOptionPathPricer instead

Class EuropeanOption(p. 330) use VanillaOption with EuropeanAnalyticEngine

Member ICGaussianRng(p. 404)**(long seed=0)** initialize with a random number generator instead.

Class McEuropean(p. 483) use VanillaOption with McEuropeanEngine

Class MultivariateAccumulator(p. 509) use SequenceStatistics instead

Class PathGenerator_old(p. 545) use PathGenerator instead

Class PathPricer_old(p. 547) use PathPricer instead

Class RandomArrayGenerator(p. 574) use RandomSequenceGenerator instead.

Class RiskMeasures(p. 583) use Statistics instead

Class RiskStatistics_old(p. 585) use IncrementalGaussianStatistics or Statistics instead

Member SimpleSwap(p. 600)**(bool payFixedRate, const Date &startDate, const Date &maturity, const Calendar &**
use the constructor taking two Schedules

Member Scheduler(p. 114) use Schedule instead

Member MakeScheduler(p. 114) use MakeSchedule instead

Member FixedRateCouponVector(p. 121)**(const std::vector< double > &nominals, const std::vector< Rate > &cou**
use the version taking a Schedule as first argument instead

Member FixedRateCouponVector(p. 121)**(const std::vector< double > &nominals, const std::vector< Rate > &cou**
use the version taking a Schedule as first argument instead

Member FixedRateCouponVector(p. 122)**(const std::vector< double > &nominals, const std::vector< Rate > &cou**
use the version taking a Schedule as first argument instead

Member FloatingRateCouponVector(p. 122)**(const std::vector< double > &nominals, const Date &startDate, cons**
use the version taking a Schedule as first argument instead

Member FloatingRateCouponVector(p. 122)**(const std::vector< double > &nominals, const Handle< Indexes::Xib**
use the version taking a Schedule as first argument instead

Member FloatingRateCouponVector(p. 123)**(const std::vector< double > &nominals, const std::vector< Spread >**
use the version taking a Schedule as first argument instead

Member IndexedCouponVector(p. 123)**(const std::vector< double > &nominals, const Date &startDate, const Da**
use the version taking a Schedule instead

Member VanillaCapFloor(p. 130) use CapFloor instead

Member VanillaCap(p. 130) use Cap instead

Member VanillaFloor(p. 130) use Floor instead

Member VanillaCollar(p. 130) use Collar instead

Index

add
    QuantLib::Math::GeneralStatistics, 380
    QuantLib::Math::IncrementalStatistics, 416
    QuantLib::RiskStatistics_old, 586
advance
    QuantLib::Calendar, 237
amount
    QuantLib::CashFlow, 252
    QuantLib::CashFlows::FixedRateCoupon, 354
    QuantLib::CashFlows::IndexedCoupon, 420
    QuantLib::CashFlows::ParCoupon, 542
    QuantLib::CashFlows::Short, 593
    QuantLib::CashFlows::SimpleCashFlow, 596
applyAfterApplying
    QuantLib::FiniteDifferences::BoundaryCondition, 224
applyAfterSolving
    QuantLib::FiniteDifferences::BoundaryCondition, 224
applyBeforeApplying
    QuantLib::FiniteDifferences::BoundaryCondition, 224
applyBeforeSolving
    QuantLib::FiniteDifferences::BoundaryCondition, 224
ARS
    QuantLib, 115
ATS
    QuantLib, 115
AUD
    QuantLib, 115
averageShortfall
    QuantLib::Math::GenericRiskStatistics, 385

BDT
    QuantLib, 115
BEF
    QuantLib, 115
BGL
    QuantLib, 115
blackVarianceImpl
    QuantLib::BlackVolatilityTermStructure, 221
blackVolImpl
    QuantLib::BlackVarianceTermStructure, 220
BoxMullerGaussianRng
    QuantLib::RandomNumbers::BoxMullerGaussianRng, 227
BRL
    QuantLib, 115
BYB
    QuantLib, 115

CAD
    QuantLib, 115
calculate
    QuantLib::Instrument, 423
    QuantLib::Patterns::LazyObject, 444
Calendar
    QuantLib::Calendar, 236
calibrate
    QuantLib::ShortRateModels::Model, 503
Character functions, 101
CHF
    QuantLib, 115
CLGaussianRng
    QuantLib::RandomNumbers::CLGaussianRng, 256
CLP
    QuantLib, 115
CNY
    QuantLib, 115
compoundForwardImpl
    QuantLib::DiscountStructure, 303
    QuantLib::ForwardRateStructure, 361
    QuantLib::TermStructures::ExtendedDiscountCurve, 341
    QuantLib::ZeroYieldStructure, 683
COP
    QuantLib, 115
Coupon
    QuantLib::CashFlows::Coupon, 277
Currency

QuantLib, 115
CYP
QuantLib, 115
CZK
QuantLib, 115

DayCounter
QuantLib::DayCounter, 293
DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE
sequencestatistics.hpp, 888
DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID
sequencestatistics.hpp, 888
DEM
QuantLib, 115
discountImpl
QuantLib::ForwardRateStructure, 361
QuantLib::ZeroYieldStructure, 683
DKK
QuantLib, 115
downsideDeviation
QuantLib::Math::GenericRiskStatistics, 384
QuantLib::Math::IncrementalStatistics, 415
downsideVariance
QuantLib::Math::GenericRiskStatistics, 383
QuantLib::Math::IncrementalStatistics, 415

EEK
QuantLib, 115
errorEstimate
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 415
errors.hpp
QL_ASSERT, 769
QL_ENSURE, 770
QL_REQUIRE, 769
EUR
QuantLib, 115
expectation
QuantLib::DiffusionProcess, 298
QuantLib::OrnsteinUhlenbeckProcess, 534
expectationValue
QuantLib::Math::GeneralStatistics, 380
expectedShortfall
QuantLib::Math::GenericRiskStatistics, 384

FixedRateCouponVector
QuantLib::CashFlows, 121
fixing
QuantLib::Index, 417
QuantLib::Indexes::Xibor, 676
FloatingRateCouponVector
QuantLib::CashFlows, 122
Following
QuantLib, 115
formula
QuantLib::BlackModel, 211
forwardImpl
QuantLib::DiscountStructure, 303
QuantLib::TermStructures::Zero-SpreadedTermStructure, 682
QuantLib::ZeroYieldStructure, 683
freeze
QuantLib::Patterns::LazyObject, 444

gaussianDownsideDeviation
QuantLib::Math::GaussianStatistics, 375
gaussianDownsideVariance
QuantLib::Math::GaussianStatistics, 375
gaussianExpectedShortfall
QuantLib::Math::GaussianStatistics, 376
QuantLib::Math::RiskMeasures, 584
gaussianPercentile
QuantLib::Math::GaussianStatistics, 376
QuantLib::Math::RiskMeasures, 583
gaussianPotentialUpside
QuantLib::Math::GaussianStatistics, 376
QuantLib::Math::RiskMeasures, 583
gaussianRegret
QuantLib::Math::GaussianStatistics, 375
gaussianValueAtRisk
QuantLib::Math::GaussianStatistics, 376
QuantLib::Math::RiskMeasures, 584
GBP
QuantLib, 115
getCovariance
QuantLib::MonteCarlo, 137
Global QuantLib macros, 95
GRD
QuantLib, 115

Handle
QuantLib::Handle, 391
History

QuantLib::History, 395
HKD
QuantLib, 115
HUF
QuantLib, 116
hypot
QuantLib::Math, 134

ICGaussianRng
QuantLib::RandomNumbers::ICGaussianRng, 404
ILS
QuantLib, 116
impliedVolatility
QuantLib::Instruments::VanillaOption, 665
QuantLib::Pricers::SingleAssetOption, 606
IndexedCouponVector
QuantLib::CashFlows, 123
Input/output functions, 102
INR
QuantLib, 116
IQD
QuantLib, 116
IRR
QuantLib, 116
isBusinessDay
QuantLib::Calendar, 236
isEndOfMonth
QuantLib::Calendar, 236
isHoliday
QuantLib::Calendar, 237
ISK
QuantLib, 116
Iterator support, 106
iteratorMacros
QL_FULL_ITERATOR_SUPPORT, 106
QL_ITERATOR, 106
QL_ITERATOR_TRAITS, 106
QL_REVERSE_ITERATOR, 106
QL_SPECIALIZE_ITERATOR_TRAITS, 106
ITL
QuantLib, 116

JoinBusinessDays
QuantLib::Calendars, 118
JoinHolidays
QuantLib::Calendars, 118
JointCalendarRule
QuantLib::Calendars, 118
JPY
QuantLib, 116

KnuthUniformRng
QuantLib::RandomNumbers::KnuthUniformRng, 437
KRW
QuantLib, 116
kurtosis
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 416
KWD
QuantLib, 116

LecuyerUniformRng
QuantLib::RandomNumbers::LecuyerUniformRng, 448
limitMacros
QL_EPSILON, 98
QL_MAX_DOUBLE, 98
QL_MAX_INT, 98
QL_MIN_DOUBLE, 98
QL_MIN_INT, 98
QL_MIN_POSITIVE_DOUBLE, 98
Link
QuantLib::Link, 455
linkTo
QuantLib::Link, 455
QuantLib::RelinkableHandle, 579
localVolImpl
QuantLib::VolTermStructures::LocalVolCurve, 460
LTL
QuantLib, 116
LVL
QuantLib, 116

macros
QL_DUMMY_RETURN, 95
QL_IO_INIT, 96
MakeScheduler
QuantLib, 114
Math functions, 97
matrixSqrt
QuantLib::Math::Matrix, 473
max
QuantLib::Math::GeneralStatistics, 380
QuantLib::Math::IncrementalStatistics, 416
mean
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 415
MersenneTwisterUniformRng

QuantLib::Random-Numbers::MersenneTwister-UniformRng, 496
min
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 416
Min and max functions, 103
ModifiedFollowing
QuantLib, 115
ModifiedPreceding
QuantLib, 115
MonthEndReference
QuantLib, 115
MTL
QuantLib, 116
MXP
QuantLib, 116

name
QuantLib::Calendar, 236
QuantLib::DayCounter, 293
QuantLib::Index, 417
QuantLib::Indexes::Xibor, 676
next
QuantLib::RandomNumbers::Knuth-UniformRng, 437
QuantLib::RandomNumbers::Lecuyer-UniformRng, 448
QuantLib::Random-Numbers::MersenneTwister-UniformRng, 496
NOK
QuantLib, 116
notifyObservers
QuantLib::Patterns::Observable, 524
NPR
QuantLib, 116
Numeric limits, 98
NZD
QuantLib, 116

operator()
QuantLib::Math::BicubicSpline-Interpolation, 191
QuantLib::Math::BilinearInterpolation, 193
QuantLib::Math::CubicSpline-Interpolation, 284
QuantLib::Math::Interpolation, 428
QuantLib::Math::Interpolation2D, 430
QuantLib::Math::LinearInterpolation, 452
QuantLib::Math::LogLinear-Interpolation, 466
operator+=
QuantLib::Math::Matrix, 473
operator==
QuantLib::Calendar, 237
QuantLib::DayCounter, 293

PathGenerator_old
QuantLib::MonteCarlo::Path-Generator_old, 545
percentile
QuantLib::Math::GeneralStatistics, 380
QuantLib::Math::RiskMeasures, 583
performCalculations
QuantLib::Instrument, 424
QuantLib::Instruments::BarrierOption, 183
QuantLib::Instruments::BinaryOption, 198
QuantLib::Instruments::Forward-VanillaOption, 366
QuantLib::Instruments::QuantoVanilla-Option, 573
QuantLib::Instruments::Stock, 615
QuantLib::Instruments::Swap, 622
QuantLib::Instruments::Swaption, 625
QuantLib::Instruments::VanillaOption, 666
QuantLib::Patterns::LazyObject, 444
PiecewiseFlatForward
QuantLib::TermStructures::Piecewise-FlatForward, 554
PKR
QuantLib, 116
PLN
QuantLib, 116
postAdjustValues
QuantLib::DiscretizedAsset, 308
QuantLib::DiscretizedOption, 311
potentialUpside
QuantLib::Math::GenericRiskStatistics, 384
QuantLib::Math::RiskMeasures, 583
preAdjustValues
QuantLib::DiscretizedAsset, 308
Preceding
QuantLib, 115
pseudoSqrt
QuantLib::Math::Matrix, 473

ql/argsandresults.hpp, 687
ql/array.hpp, 688
ql/blackmodel.hpp, 689

ql/calendar.cpp, 690
ql/calendar.hpp, 691
ql/Calendars/budapest.cpp, 692
ql/Calendars/budapest.hpp, 693
ql/Calendars/frankfurt.cpp, 694
ql/Calendars/frankfurt.hpp, 695
ql/Calendars/helsinki.cpp, 696
ql/Calendars/helsinki.hpp, 697
ql/Calendars/johannesburg.cpp, 698
ql/Calendars/johannesburg.hpp, 699
ql/Calendars/jointcalendar.cpp, 700
ql/Calendars/jointcalendar.hpp, 701
ql/Calendars/london.cpp, 702
ql/Calendars/london.hpp, 703
ql/Calendars/milan.cpp, 704
ql/Calendars/milan.hpp, 705
ql/Calendars/newyork.cpp, 706
ql/Calendars/newyork.hpp, 707
ql/Calendars/nullcalendar.hpp, 708
ql/Calendars/oslo.cpp, 709
ql/Calendars/oslo.hpp, 710
ql/Calendars/stockholm.cpp, 711
ql/Calendars/stockholm.hpp, 712
ql/Calendars/sydney.cpp, 713
ql/Calendars/sydney.hpp, 714
ql/Calendars/target.cpp, 715
ql/Calendars/target.hpp, 716
ql/Calendars/tokyo.cpp, 717
ql/Calendars/tokyo.hpp, 718
ql/Calendars/toronto.cpp, 719
ql/Calendars/toronto.hpp, 720
ql/Calendars/warsaw.cpp, 721
ql/Calendars/warsaw.hpp, 722
ql/Calendars/wellington.cpp, 723
ql/Calendars/wellington.hpp, 724
ql/Calendars/zurich.cpp, 725
ql/Calendars/zurich.hpp, 726
ql/capvolstructures.hpp, 727
ql/cashflow.hpp, 728
ql/CashFlows/basispointsensitivity.cpp, 729
ql/CashFlows/basispointsensitivity.hpp, 730
ql/CashFlows/cashflowvectors.cpp, 731
ql/CashFlows/cashflowvectors.hpp, 732
ql/CashFlows/coupon.hpp, 733
ql/CashFlows/fixedratecoupon.hpp, 734
ql/CashFlows/floatingratecoupon.hpp, 735
ql/CashFlows/inarrearindexedcoupon.hpp, 736
ql/CashFlows/indexcashflowvectors.hpp, 737
ql/CashFlows/indexedcoupon.hpp, 738
ql/CashFlows/parcoupon.cpp, 739
ql/CashFlows/parcoupon.hpp, 740
ql/CashFlows/shortfloatingcoupon.cpp, 741
ql/CashFlows/shortfloatingcoupon.hpp, 742
ql/CashFlows/shortindexedcoupon.hpp, 743
ql/CashFlows/simplecashflow.hpp, 744
ql/CashFlows/timebasket.cpp, 745
ql/CashFlows/timebasket.hpp, 746
ql/CashFlows/upfrontindexedcoupon.hpp, 747
ql/currency.hpp, 748
ql/dataformatters.cpp, 749
ql/dataformatters.hpp, 750
ql/dataparsers.cpp, 751
ql/dataparsers.hpp, 752
ql/date.cpp, 753
ql/date.hpp, 754
ql/daycounter.hpp, 755
ql/DayCounters/actual360.hpp, 756
ql/DayCounters/actual365.hpp, 757
ql/DayCounters/actualactual.cpp, 758
ql/DayCounters/actualactual.hpp, 759
ql/DayCounters/simpledaycounter.cpp, 760
ql/DayCounters/simpledaycounter.hpp, 761
ql/DayCounters/thirty360.cpp, 762
ql/DayCounters/thirty360.hpp, 763
ql/diffusionprocess.cpp, 764
ql/diffusionprocess.hpp, 765
ql/discretizedasset.cpp, 766
ql/discretizedasset.hpp, 767
ql/disposable.hpp, 768
ql/errors.hpp, 769
ql/exercise.cpp, 771
ql/exercise.hpp, 772
ql/FiniteDifferences/americancondition.hpp, 773
ql/FiniteDifferences/boundarycondition.cpp, 774
ql/FiniteDifferences/boundarycondition.hpp, 775
ql/FiniteDifferences/bsmoperator.cpp, 776
ql/FiniteDifferences/bsmoperator.hpp, 777
ql/FiniteDifferences/cranknicolson.hpp, 778
ql/FiniteDifferences/dminus.hpp, 779
ql/FiniteDifferences/dplus.hpp, 780
ql/FiniteDifferences/dplusdminus.hpp, 781
ql/FiniteDifferences/dzero.hpp, 782
ql/FiniteDifferences/expliciteuler.hpp, 783
ql/FiniteDifferences/fdtypedefs.hpp, 784
ql/FiniteDifferences/finitedifferencemodel.hpp, 785
ql/FiniteDifferences/impliciteuler.hpp, 786
ql/FiniteDifferences/mixedscheme.hpp, 787
ql/FiniteDifferences/onefactoroperator.cpp, 788
ql/FiniteDifferences/onefactoroperator.hpp, 789

ql/FiniteDifferences/shoutcondition.hpp, 790
ql/FiniteDifferences/stepcondition.hpp, 791
ql/FiniteDifferences/tridiagonaloperator.cpp, 792
ql/FiniteDifferences/tridiagonaloperator.hpp, 793
ql/FiniteDifferences/valueatcenter.cpp, 794
ql/FiniteDifferences/valueatcenter.hpp, 795
ql/functions/daycounters.cpp, 796
ql/functions/daycounters.hpp, 797
ql/functions/mathf.cpp, 798
ql/functions/mathf.hpp, 799
ql/functions/vols.cpp, 800
ql/functions/vols.hpp, 801
ql/grid.cpp, 802
ql/grid.hpp, 803
ql/handle.hpp, 804
ql/history.hpp, 805
ql/index.hpp, 806
ql/Indexes/audlibor.hpp, 807
ql/Indexes/cadlibor.hpp, 808
ql/Indexes/chflibor.hpp, 809
ql/Indexes/euribor.hpp, 810
ql/Indexes/gbplibor.hpp, 811
ql/Indexes/jpylibor.hpp, 812
ql/Indexes/usdlibor.hpp, 813
ql/Indexes/xibor.cpp, 814
ql/Indexes/xibor.hpp, 815
ql/Indexes/xibormanager.cpp, 816
ql/Indexes/xibormanager.hpp, 817
ql/Indexes/zarlibor.hpp, 818
ql/instrument.hpp, 819
ql/Instruments/barrieroption.cpp, 820
ql/Instruments/barrieroption.hpp, 821
ql/Instruments/binaryoption.cpp, 822
ql/Instruments/binaryoption.hpp, 823
ql/Instruments/capfloor.cpp, 824
ql/Instruments/capfloor.hpp, 825
ql/Instruments/forwardvanillaoption.cpp, 826
ql/Instruments/forwardvanillaoption.hpp, 827
ql/Instruments/quantoforwardvanillaoption.cpp, 828
ql/Instruments/quantoforwardvanillaoption.hpp, 829
ql/Instruments/quantovanillaoption.cpp, 830
ql/Instruments/quantovanillaoption.hpp, 831
ql/Instruments/simpleswap.cpp, 832
ql/Instruments/simpleswap.hpp, 833
ql/Instruments/stock.cpp, 834
ql/Instruments/stock.hpp, 835
ql/Instruments/swap.cpp, 836
ql/Instruments/swap.hpp, 837
ql/Instruments/swaption.cpp, 838
ql/Instruments/swaption.hpp, 839
ql/Instruments/vanillaoption.cpp, 840
ql/Instruments/vanillaoption.hpp, 841
ql/Lattices/binomialtree.cpp, 842
ql/Lattices/binomialtree.hpp, 843
ql/Lattices/bsmlattice.cpp, 844
ql/Lattices/bsmlattice.hpp, 845
ql/Lattices/lattice.cpp, 846
ql/Lattices/lattice.hpp, 847
ql/Lattices/lattice2d.cpp, 848
ql/Lattices/lattice2d.hpp, 849
ql/Lattices/tree.hpp, 850
ql/Lattices/trinomialtree.cpp, 851
ql/Lattices/trinomialtree.hpp, 852
ql/marketelement.hpp, 853
ql/Math/bicubicsplineinterpolation.hpp, 854
ql/Math/bilinearinterpolation.hpp, 855
ql/Math/chisquaredistribution.cpp, 856
ql/Math/chisquaredistribution.hpp, 857
ql/Math/cubicspline.hpp, 858
ql/Math/discrepancystatistics.cpp, 859
ql/Math/discrepancystatistics.hpp, 860
ql/Math/errorfunction.cpp, 861
ql/Math/errorfunction.hpp, 862
ql/Math/functional.hpp, 863
ql/Math/gammadistribution.cpp, 864
ql/Math/gammadistribution.hpp, 865
ql/Math/gaussianstatistics.hpp, 866
ql/Math/generalstatistics.cpp, 867
ql/Math/generalstatistics.hpp, 868
ql/Math/incrementalstatistics.cpp, 869
ql/Math/incrementalstatistics.hpp, 870
ql/Math/interpolation.hpp, 871
ql/Math/interpolation2D.hpp, 872
ql/Math/interpolationtraits.hpp, 873
ql/Math/kronrodintegral.hpp, 874
ql/Math/lexicographicalview.hpp, 875
ql/Math/linearinterpolation.hpp, 876
ql/Math/loglinearinterpolation.hpp, 877
ql/Math/matrix.cpp, 878
ql/Math/matrix.hpp, 879
ql/Math/multivariateaccumulator.cpp, 880
ql/Math/multivariateaccumulator.hpp, 881
ql/Math/normaldistribution.cpp, 882
ql/Math/normaldistribution.hpp, 883
ql/Math/primenumbers.cpp, 884
ql/Math/primenumbers.hpp, 885
ql/Math/riskmeasures.hpp, 886
ql/Math/segmentintegral.hpp, 887
ql/Math/sequencestatistics.hpp, 888

ql/Math/statistics.hpp, 890
ql/Math/svd.cpp, 891
ql/Math/svd.hpp, 892
ql/Math/symmetriceigenvalues.hpp, 893
ql/Math/symmetricschurdecomposition.cpp, 894
ql/Math/symmetricschurdecomposition.hpp, 895
ql/Math/trapezoidintegral.hpp, 896
ql/MonteCarlo/arithmeticapopathpricer.cpp, 897
ql/MonteCarlo/arithmeticapopathpricer.hpp, 898
ql/MonteCarlo/arithmeticasopathpricer.cpp, 899
ql/MonteCarlo/arithmeticasopathpricer.hpp, 900
ql/MonteCarlo/barrierpathpricer.cpp, 901
ql/MonteCarlo/barrierpathpricer.hpp, 902
ql/MonteCarlo/basketpathpricer.cpp, 903
ql/MonteCarlo/basketpathpricer.hpp, 904
ql/MonteCarlo/binarypathpricer.cpp, 905
ql/MonteCarlo/binarypathpricer.hpp, 906
ql/MonteCarlo/brownianbridge.hpp, 907
ql/MonteCarlo/cliquetoptionpathpricer.cpp, 908
ql/MonteCarlo/cliquetoptionpathpricer.hpp, 909
ql/MonteCarlo/europeanpathpricer.cpp, 910
ql/MonteCarlo/europeanpathpricer.hpp, 911
ql/MonteCarlo/everestpathpricer.cpp, 912
ql/MonteCarlo/everestpathpricer.hpp, 913
ql/MonteCarlo/geometricapopathpricer.cpp, 914
ql/MonteCarlo/geometricapopathpricer.hpp, 915
ql/MonteCarlo/geometricasopathpricer.cpp, 916
ql/MonteCarlo/geometricasopathpricer.hpp, 917
ql/MonteCarlo/getcovariance.hpp, 918
ql/MonteCarlo/himalayapathpricer.cpp, 919
ql/MonteCarlo/himalayapathpricer.hpp, 920
ql/MonteCarlo/maxbasketpathpricer.cpp, 921
ql/MonteCarlo/maxbasketpathpricer.hpp, 922
ql/MonteCarlo/mctraits.hpp, 923
ql/MonteCarlo/mctypedefs.hpp, 924
ql/MonteCarlo/montecarlomodel.hpp, 925
ql/MonteCarlo/multipath.hpp, 926
ql/MonteCarlo/multipathgenerator.hpp, 927
ql/MonteCarlo/pagodapathpricer.cpp, 928
ql/MonteCarlo/pagodapathpricer.hpp, 929
ql/MonteCarlo/path.hpp, 930
ql/MonteCarlo/pathgenerator.hpp, 931
ql/MonteCarlo/pathpricer.hpp, 932
ql/MonteCarlo/performanceoptionpathpricer.cpp, 933
ql/MonteCarlo/performanceoptionpathpricer.hpp, 934
ql/MonteCarlo/sample.hpp, 935
ql/null.hpp, 936
ql/numericalmethod.hpp, 937
ql/Optimization/armijo.cpp, 938
ql/Optimization/armijo.hpp, 939
ql/Optimization/conjugategradient.cpp, 940
ql/Optimization/conjugategradient.hpp, 941
ql/Optimization/constraint.hpp, 942
ql/Optimization/costfunction.hpp, 943
ql/Optimization/criteria.hpp, 944
ql/Optimization/leastsquare.hpp, 945
ql/Optimization/linesearch.hpp, 946
ql/Optimization/method.hpp, 947
ql/Optimization/problem.hpp, 948
ql/Optimization/simplex.cpp, 949
ql/Optimization/simplex.hpp, 950
ql/Optimization/steepestdescent.cpp, 951
ql/Optimization/steepestdescent.hpp, 952
ql/option.hpp, 953
ql/Patterns/bridge.hpp, 954
ql/Patterns/curiouslyrecurring.hpp, 955
ql/Patterns/lazyobject.hpp, 956
ql/Patterns/observable.hpp, 957
ql/Patterns/visitor.hpp, 958
ql/payoff.hpp, 959
ql/Pricers/analyticalcapfloor.cpp, 960
ql/Pricers/analyticalcapfloor.hpp, 961
ql/Pricers/barrieroptionpricer.cpp, 962
ql/Pricers/barrieroptionpricer.hpp, 963
ql/Pricers/binaryoptionpricer.cpp, 964
ql/Pricers/binaryoptionpricer.hpp, 965
ql/Pricers/blackcapfloor.cpp, 966
ql/Pricers/blackcapfloor.hpp, 967
ql/Pricers/blackswaption.cpp, 968
ql/Pricers/blackswaption.hpp, 969
ql/Pricers/capfloorpricer.cpp, 970
ql/Pricers/capfloorpricer.hpp, 971
ql/Pricers/cliquetoption.cpp, 972
ql/Pricers/continuousgeometricapo.hpp, 973
ql/Pricers/discretegeometricapo.cpp, 974
ql/Pricers/discretegeometricapo.hpp, 975
ql/Pricers/discretegeometricaso.cpp, 976
ql/Pricers/discretegeometricaso.hpp, 977
ql/Pricers/europeanoption.cpp, 978
ql/Pricers/europeanoption.hpp, 979
ql/Pricers/fdamericanoption.hpp, 980
ql/Pricers/fdbermudanoption.cpp, 981

ql/Pricers/fdbermudanoption.hpp, 982
ql/Pricers/fdbsmoption.cpp, 983
ql/Pricers/fdbsmoption.hpp, 984
ql/Pricers/fddividendamericanoption.cpp, 985
ql/Pricers/fddividendamericanoption.hpp, 986
ql/Pricers/fddividendeuropeanoption.cpp, 987
ql/Pricers/fddividendeuropeanoption.hpp, 988
ql/Pricers/fddividendoption.cpp, 989
ql/Pricers/fddividendoption.hpp, 990
ql/Pricers/fddividendshoutoption.cpp, 991
ql/Pricers/fddividendshoutoption.hpp, 992
ql/Pricers/fdeuropean.cpp, 993
ql/Pricers/fdeuropean.hpp, 994
ql/Pricers/fdmultiperiodoption.cpp, 995
ql/Pricers/fdmultiperiodoption.hpp, 996
ql/Pricers/fdshoutoption.hpp, 997
ql/Pricers/fdstepconditionoption.cpp, 998
ql/Pricers/fdstepconditionoption.hpp, 999
ql/Pricers/jamshidianswaption.cpp, 1000
ql/Pricers/jamshidianswaption.hpp, 1001
ql/Pricers/mcbasket.cpp, 1002
ql/Pricers/mcbasket.hpp, 1003
ql/Pricers/mccliquetoption.cpp, 1004
ql/Pricers/mccliquetoption.hpp, 1005
ql/Pricers/mcdiscretearithmeticapo.cpp, 1006
ql/Pricers/mcdiscretearithmeticapo.hpp, 1007
ql/Pricers/mcdiscretearithmeticaso.cpp, 1008
ql/Pricers/mcdiscretearithmeticaso.hpp, 1009
ql/Pricers/mceuropean.cpp, 1010
ql/Pricers/mceuropean.hpp, 1011
ql/Pricers/mceverest.cpp, 1012
ql/Pricers/mceverest.hpp, 1013
ql/Pricers/mchimalaya.cpp, 1014
ql/Pricers/mchimalaya.hpp, 1015
ql/Pricers/mcmaxbasket.cpp, 1016
ql/Pricers/mcmaxbasket.hpp, 1017
ql/Pricers/mcpagoda.cpp, 1018
ql/Pricers/mcpagoda.hpp, 1019
ql/Pricers/mcperformanceoption.cpp, 1020
ql/Pricers/mcperformanceoption.hpp, 1021
ql/Pricers/mcpricer.hpp, 1022
ql/Pricers/performanceoption.cpp, 1023
ql/Pricers/performanceoption.hpp, 1024
ql/Pricers/singleassetoption.cpp, 1025
ql/Pricers/singleassetoption.hpp, 1026
ql/Pricers/swaptionpricer.cpp, 1027
ql/Pricers/swaptionpricer.hpp, 1028
ql/Pricers/treecapfloor.cpp, 1029
ql/Pricers/treecapfloor.hpp, 1030
ql/Pricers/treeswaption.cpp, 1031
ql/Pricers/treeswaption.hpp, 1032
ql/pricingengine.hpp, 1033
ql/PricingEngines/americanmcengines.cpp, 1034
ql/PricingEngines/americanmcengines.hpp, 1035
ql/PricingEngines/analyticamericanbinaryengine.cpp, 1036
ql/PricingEngines/analyticeuropeanbinaryengine.cpp, 1037
ql/PricingEngines/analyticeuropeanengine.cpp, 1038
ql/PricingEngines/barrierengines.hpp, 1039
ql/PricingEngines/binaryengines.hpp, 1040
ql/PricingEngines/binomialvanillaengine.cpp, 1041
ql/PricingEngines/cliquetengines.hpp, 1042
ql/PricingEngines/discretizedvanillaoption.cpp, 1043
ql/PricingEngines/discretizedvanillaoption.hpp, 1044
ql/PricingEngines/forwardengines.hpp, 1045
ql/PricingEngines/genericengine.hpp, 1046
ql/PricingEngines/integralengines.cpp, 1047
ql/PricingEngines/latticeshortratemodelengine.hpp, 1048
ql/PricingEngines/mcengine.hpp, 1049
ql/PricingEngines/quantoengines.hpp, 1050
ql/PricingEngines/vanillaengines.hpp, 1051
ql/qldefines.hpp, 1052
ql/RandomNumbers/boxmullergaussianrng.hpp, 1054
ql/RandomNumbers/centrallimitgaussianrng.hpp, 1055
ql/RandomNumbers/haltonrsg.cpp, 1056
ql/RandomNumbers/haltonrsg.hpp, 1057
ql/RandomNumbers/inversecumgaussianrng.hpp, 1058
ql/RandomNumbers/inversecumgaussianrsg.hpp, 1059
ql/RandomNumbers/knuthuniformrng.cpp, 1060
ql/RandomNumbers/knuthuniformrng.hpp, 1061
ql/RandomNumbers/lecuyeruniformrng.cpp, 1062
ql/RandomNumbers/lecuyeruniformrng.hpp, 1063

ql/RandomNumbers/mt19937uniformrng.cpp, 1064
ql/RandomNumbers/mt19937uniformrng.hpp, 1065
ql/RandomNumbers/randomarraygenerator.hpp, 1066
ql/RandomNumbers/randomsequencegenerator.hpp, 1067
ql/RandomNumbers/rngtypedefs.hpp, 1068
ql/RandomNumbers/sobolrsg.cpp, 1069
ql/RandomNumbers/sobolrsg.hpp, 1070
ql/relinkablehandle.hpp, 1071
ql/scheduler.cpp, 1072
ql/scheduler.hpp, 1073
ql/ShortRateModels/calibrationhelper.cpp, 1074
ql/ShortRateModels/calibrationhelper.hpp, 1075
ql/ShortRateModels/CalibrationHelpers/caphelper.cpp, 1076
ql/ShortRateModels/CalibrationHelpers/caphelper.hpp, 1077
ql/ShortRateModels/CalibrationHelpers/swaptionhelper.cpp, 1078
ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp, 1079
ql/ShortRateModels/model.cpp, 1080
ql/ShortRateModels/model.hpp, 1081
ql/ShortRateModels/onefactormodel.cpp, 1082
ql/ShortRateModels/onefactormodel.hpp, 1083
ql/ShortRateModels/OneFactorModels/blackkarasinski.cpp, 1084
ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp, 1085
ql/ShortRateModels/OneFactorModels/coxingersollross.cpp, 1086
ql/ShortRateModels/OneFactorModels/coxingersollross.hpp, 1087
ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.cpp, 1088
ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp, 1089
ql/ShortRateModels/OneFactorModels/hullwhite.cpp, 1090
ql/ShortRateModels/OneFactorModels/hullwhite.hpp, 1091
ql/ShortRateModels/OneFactorModels/vasicek.cpp, 1092
ql/ShortRateModels/OneFactorModels/vasicek.hpp, 1093
ql/ShortRateModels/parameter.hpp, 1094
ql/ShortRateModels/twofactormodel.cpp, 1095
ql/ShortRateModels/twofactormodel.hpp, 1096
ql/ShortRateModels/TwoFactorModels/g2.cpp, 1097
ql/ShortRateModels/TwoFactorModels/g2.hpp, 1098
ql/solver1d.hpp, 1099
ql/Solvers1D/bisection.hpp, 1100
ql/Solvers1D/brent.hpp, 1101
ql/Solvers1D/falseposition.hpp, 1102
ql/Solvers1D/newton.hpp, 1103
ql/Solvers1D/newtonsafe.hpp, 1104
ql/Solvers1D/ridder.hpp, 1105
ql/Solvers1D/secant.hpp, 1106
ql/swaptionvolstructure.hpp, 1107
ql/termstructure.hpp, 1108
ql/TermStructures/affinetermstructure.cpp, 1109
ql/TermStructures/affinetermstructure.hpp, 1110
ql/TermStructures/compoundforward.cpp, 1111
ql/TermStructures/compoundforward.hpp, 1112
ql/TermStructures/discountcurve.cpp, 1113
ql/TermStructures/discountcurve.hpp, 1114
ql/TermStructures/drifttermstructure.hpp, 1115
ql/TermStructures/extendeddiscountcurve.cpp, 1116
ql/TermStructures/extendeddiscountcurve.hpp, 1117
ql/TermStructures/flatforward.hpp, 1118
ql/TermStructures/forwardspreadedtermstructure.hpp, 1119
ql/TermStructures/impliedtermstructure.hpp, 1120
ql/TermStructures/piecewiseflatforward.cpp, 1121
ql/TermStructures/piecewiseflatforward.hpp, 1122
ql/TermStructures/quantotermstructure.hpp, 1123
ql/TermStructures/ratehelpers.cpp, 1124
ql/TermStructures/ratehelpers.hpp, 1125
ql/TermStructures/zerocurve.cpp, 1126
ql/TermStructures/zerocurve.hpp, 1127
ql/TermStructures/zerospreadedtermstructure.hpp, 1128
ql/types.hpp, 1129
ql/Utilities/combiningiterator.hpp, 1130
ql/Utilities/couplingiterator.hpp, 1131

ql/Utilities/filteringiterator.hpp, 1132
ql/Utilities/iteratorcategories.hpp, 1133
ql/Utilities/processingiterator.hpp, 1134
ql/Utilities/steppingiterator.hpp, 1135
ql/Volatilities/blackconstantvol.hpp, 1136
ql/Volatilities/blackvariancecurve.cpp, 1137
ql/Volatilities/blackvariancecurve.hpp, 1138
ql/Volatilities/blackvariancesurface.cpp, 1139
ql/Volatilities/blackvariancesurface.hpp, 1140
ql/Volatilities/capflatvolvector.hpp, 1141
ql/Volatilities/impliedvoltermstructure.hpp, 1142
ql/Volatilities/localconstantvol.hpp, 1143
ql/Volatilities/localvolcurve.hpp, 1144
ql/Volatilities/localvolsurface.cpp, 1145
ql/Volatilities/localvolsurface.hpp, 1146
ql/Volatilities/swaptionvolmatrix.hpp, 1147
ql/voltermstructure.cpp, 1148
ql/voltermstructure.hpp, 1149
QL_ALLOW_TEMPLATE_METHOD_CALLS
templateMacros, 104
QL_ASSERT
errors.hpp, 769
QL_DECLARE_TEMPLATE_SPECIALIZATIONS
templateMacros, 104
QL_DUMMY_RETURN
macros, 95
QL_ENSURE
errors.hpp, 770
QL_EPSILON
limitMacros, 98
QL_FULL_ITERATOR_SUPPORT
iteratorMacros, 106
QL_IO_INIT
macros, 96
QL_ITERATOR
iteratorMacros, 106
QL_ITERATOR_TRAITS
iteratorMacros, 106
QL_MAX_DOUBLE
limitMacros, 98
QL_MAX_INT
limitMacros, 98
QL_MIN_DOUBLE
limitMacros, 98
QL_MIN_INT
limitMacros, 98
QL_MIN_POSITIVE_DOUBLE
limitMacros, 98
QL_REQUIRE
errors.hpp, 769
QL_REVERSE_ITERATOR
iteratorMacros, 106
QL_SPECIALIZE_ITERATOR_TRAITS
iteratorMacros, 106
QL_TEMPLATE_METAPROGRAMMING_WORKS
templateMacros, 104
QL_TYPENAME
templateMacros, 104
QuantLib, 107
ARS, 115
ATS, 115
AUD, 115
BDT, 115
BEF, 115
BGL, 115
BRL, 115
BYB, 115
CAD, 115
CHF, 115
CLP, 115
CNY, 115
COP, 115
CYP, 115
CZK, 115
DEM, 115
DKK, 115
EEK, 115
EUR, 115
Following, 115
GBP, 115
GRD, 115
HKD, 115
HUF, 116
ILS, 116
INR, 116
IQD, 116
IRR, 116
ISK, 116
ITL, 116
JPY, 116
KRW, 116
KWD, 116
LTL, 116
LVL, 116
ModifiedFollowing, 115
ModifiedPreceding, 115
MonthEndReference, 115
MTL, 116
MXP, 116
NOK, 116
NPR, 116
NZD, 116

PKR, 116
PLN, 116
Preceding, 115
ROL, 116
SAR, 116
SEK, 116
SGD, 116
SIT, 116
SKK, 116
THB, 116
TRL, 116
TTD, 116
TWD, 116
USD, 116
VEB, 116
ZAR, 116
QuantLib
Currency, 115
MakeScheduler, 114
RollingConvention, 115
Scheduler, 114
Weekday, 116
QuantLib::AmericanExercise, 162
QuantLib::Arguments, 169
QuantLib::Array, 173
QuantLib::ArrayFormatter, 176
QuantLib::AssertionFailedError, 177
QuantLib::AssetOrNothingPayoff, 178
QuantLib::Barrier, 180
QuantLib::BermudanExercise, 189
QuantLib::Binary, 195
QuantLib::BlackModel, 211
QuantLib::BlackModel
formula, 211
update, 211
QuantLib::BlackScholesProcess, 214
QuantLib::BlackVarianceTermStructure, 220
QuantLib::BlackVarianceTermStructure
blackVolImpl, 220
QuantLib::BlackVolatilityTermStructure, 221
QuantLib::BlackVolatilityTermStructure
blackVarianceImpl, 221
QuantLib::BlackVolTermStructure, 222
QuantLib::Calendar, 235
QuantLib::Calendar
advance, 237
Calendar, 236
isBusinessDay, 236
isEndOfMonth, 236
isHoliday, 237
name, 236
operator==, 237
roll, 237
QuantLib::Calendar::WesternImpl, 238
QuantLib::CalendarImpl, 239
QuantLib::Calendars, 117
JoinBusinessDays, 118
JoinHolidays, 118
QuantLib::Calendars
JointCalendarRule, 118
QuantLib::Calendars::Budapest, 233
QuantLib::Calendars::Frankfurt, 367
QuantLib::Calendars::Helsinki, 392
QuantLib::Calendars::Johannesburg, 434
QuantLib::Calendars::JointCalendar, 435
QuantLib::Calendars::London, 467
QuantLib::Calendars::Milan, 499
QuantLib::Calendars::NewYork, 514
QuantLib::Calendars::NullCalendar, 520
QuantLib::Calendars::Oslo, 536
QuantLib::Calendars::Stockholm, 616
QuantLib::Calendars::Sydney, 631
QuantLib::Calendars::TARGET, 633
QuantLib::Calendars::Tokyo, 643
QuantLib::Calendars::Toronto, 645
QuantLib::Calendars::Warsaw, 673
QuantLib::Calendars::Wellington, 674
QuantLib::Calendars::Zurich, 685
QuantLib::CapFlatVolatilityStructure, 244
QuantLib::CapletForwardVolatilityStructure, 251
QuantLib::CashFlow, 252
QuantLib::CashFlow
amount, 252
QuantLib::CashFlows, 119
QuantLib::CashFlows
FixedRateCouponVector, 121
FloatingRateCouponVector, 122
IndexedCouponVector, 123
QuantLib::CashFlows::BPSCalculator, 228
QuantLib::CashFlows::Coupon, 276
QuantLib::CashFlows::Coupon
Coupon, 277
QuantLib::CashFlows::FixedRateCoupon, 353
QuantLib::CashFlows::FixedRateCoupon
amount, 354
QuantLib::CashFlows::FloatingRateCoupon, 355
QuantLib::CashFlows::InArrearIndexedCoupon, 413
QuantLib::CashFlows::IndexedCoupon, 419
QuantLib::CashFlows::IndexedCoupon
amount, 420
update, 420
QuantLib::CashFlows::ParCoupon, 541
QuantLib::CashFlows::ParCoupon
amount, 542

update, 542
QuantLib::CashFlows::Short, 593
QuantLib::CashFlows::Short
amount, 593
QuantLib::CashFlows::ShortFloatingRateCoupon, 594
QuantLib::CashFlows::SimpleCashFlow, 596
QuantLib::CashFlows::SimpleCashFlow
amount, 596
QuantLib::CashFlows::TimeBasket, 641
QuantLib::CashFlows::UpFrontIndexedCoupon, 660
QuantLib::CashOrNothingPayoff, 254
QuantLib::CompositeMarketElement, 267
QuantLib::CompositeMarketElement
update, 267
QuantLib::CurrencyFormatter, 287
QuantLib::Date, 288
QuantLib::DateFormatter, 291
QuantLib::DayCounter, 292
QuantLib::DayCounter
DayCounter, 293
name, 293
operator==, 293
QuantLib::DayCounterImpl, 294
QuantLib::DayCounters, 124
QuantLib::DayCounters::Actual360, 153
QuantLib::DayCounters::Actual365, 154
QuantLib::DayCounters::ActualActual, 155
QuantLib::DayCounters::SimpleDayCounter, 597
QuantLib::DayCounters::Thirty360, 639
QuantLib::DerivedMarketElement, 297
QuantLib::DerivedMarketElement
update, 297
QuantLib::DiffusionProcess, 298
QuantLib::DiffusionProcess
expectation, 298
variance, 298
QuantLib::DiscountStructure, 303
QuantLib::DiscountStructure
compoundForwardImpl, 303
forwardImpl, 303
zeroYieldImpl, 303
QuantLib::DiscretizedAsset, 308
QuantLib::DiscretizedAsset
postAdjustValues, 308
preAdjustValues, 308
QuantLib::DiscretizedDiscountBond, 310
QuantLib::DiscretizedOption, 311
QuantLib::DiscretizedOption
postAdjustValues, 311
QuantLib::Disposable, 313
QuantLib::DoubleFormatter, 315
QuantLib::Error, 325
QuantLib::EuroFormatter, 328
QuantLib::EuropeanExercise, 329
QuantLib::Exercise, 334
QuantLib::FiniteDifferences, 125
QuantLib::FiniteDifferences::AmericanCondition, 161
QuantLib::FiniteDifferences::BoundaryCondition, 224
QuantLib::FiniteDifferences::BoundaryCondition
applyAfterApplying, 224
applyAfterSolving, 224
applyBeforeApplying, 224
applyBeforeSolving, 224
setTime, 225
Side, 224
QuantLib::FiniteDifferences::BSMOperator, 232
QuantLib::FiniteDifferences::CrankNicolson, 282
QuantLib::FiniteDifferences::DirichletBC, 300
QuantLib::FiniteDifferences::DirichletBC
setTime, 300
QuantLib::FiniteDifferences::DMinus, 314
QuantLib::FiniteDifferences::DPlus, 316
QuantLib::FiniteDifferences::DPlusDMinus, 317
QuantLib::FiniteDifferences::DZero, 320
QuantLib::FiniteDifferences::ExplicitEuler, 335
QuantLib::FiniteDifferences::FiniteDifferenceModel, 352
QuantLib::FiniteDifferences::FiniteDifferenceModel
rollback, 352
QuantLib::FiniteDifferences::ImplicitEuler, 408
QuantLib::FiniteDifferences::MixedScheme, 500
QuantLib::FiniteDifferences::NeumannBC, 511
QuantLib::FiniteDifferences::NeumannBC
setTime, 511
QuantLib::FiniteDifferences::OneFactorOperator, 532
QuantLib::FiniteDifferences::ShoutCondition, 595
QuantLib::FiniteDifferences::StepCondition, 612
QuantLib::FiniteDifferences::TridiagonalOperator, 651

QuantLib::FiniteDifferences::TridiagonalOperator::TimeSetter, 653
QuantLib::ForwardRateStructure, 361
QuantLib::ForwardRateStructure
compoundForwardImpl, 361
discountImpl, 361
zeroYieldImpl, 361
QuantLib::Greeks, 388
QuantLib::Handle, 390
QuantLib::Handle
Handle, 391
QuantLib::History, 394
QuantLib::History
History, 395
QuantLib::History::const_iterator, 397
QuantLib::History::Entry, 399
QuantLib::IllegalArgumentError, 406
QuantLib::IllegalResultError, 407
QuantLib::Index, 417
QuantLib::Index
fixing, 417
name, 417
QuantLib::IndexError, 421
QuantLib::Indexes, 127
QuantLib::Indexes::AUDLibor, 179
QuantLib::Indexes::CADLibor, 234
QuantLib::Indexes::CHFLibor, 255
QuantLib::Indexes::Euribor, 327
QuantLib::Indexes::GBPLibor, 377
QuantLib::Indexes::JPYLibor, 436
QuantLib::Indexes::USDLibor, 661
QuantLib::Indexes::Xibor, 675
QuantLib::Indexes::Xibor
fixing, 676
name, 676
update, 676
QuantLib::Indexes::XiborManager, 677
QuantLib::Indexes::ZARLibor, 678
QuantLib::Instrument, 422
QuantLib::Instrument
calculate, 423
performCalculations, 424
setPricingEngine, 423
setupArguments, 423
setupExpired, 423
QuantLib::Instruments, 128
QuantLib::Instruments
VanillaCap, 130
VanillaCapFloor, 130
VanillaCollar, 130
VanillaFloor, 130
QuantLib::Instruments::BarrierOption, 182
QuantLib::Instruments::BarrierOption
performCalculations, 183
setupArguments, 183
setupExpired, 183
QuantLib::Instruments::BarrierOption::arguments, 185
QuantLib::Instruments::BarrierOption::results, 186
QuantLib::Instruments::BinaryOption, 197
QuantLib::Instruments::BinaryOption
performCalculations, 198
setupArguments, 198
setupExpired, 198
QuantLib::Instruments::BinaryOption::arguments, 200
QuantLib::Instruments::BinaryOption::results, 201
QuantLib::Instruments::Cap, 243
QuantLib::Instruments::CapFloor, 247
QuantLib::Instruments::CapFloor
setupArguments, 248
QuantLib::Instruments::CapFloor::arguments, 249
QuantLib::Instruments::CapFloor::results, 250
QuantLib::Instruments::CliquetOption, 258
QuantLib::Instruments::CliquetOption::arguments, 260
QuantLib::Instruments::Collar, 263
QuantLib::Instruments::Floor, 357
QuantLib::Instruments::ForwardOptionArguments, 359
QuantLib::Instruments::ForwardVanillaOption, 365
QuantLib::Instruments::ForwardVanilla-Option
performCalculations, 366
setupArguments, 366
QuantLib::Instruments::QuantoForwardVanillaOption, 566
QuantLib::Instruments::QuantoForward-VanillaOption
setupArguments, 567
QuantLib::Instruments::QuantoOptionArguments, 568
QuantLib::Instruments::QuantoOptionResults, 569
QuantLib::Instruments::QuantoVanillaOption, 572
QuantLib::Instruments::QuantoVanilla-Option
performCalculations, 573
setupArguments, 573
setupExpired, 573
QuantLib::Instruments::SimpleSwap, 599
QuantLib::Instruments::SimpleSwap

setupArguments, 600
SimpleSwap, 600
QuantLib::Instruments::SimpleSwap::arguments, 601
QuantLib::Instruments::SimpleSwap::results, 602
QuantLib::Instruments::Stock, 615
QuantLib::Instruments::Stock
performCalculations, 615
QuantLib::Instruments::Swap, 621
QuantLib::Instruments::Swap
performCalculations, 622
sensitivity, 622
setupExpired, 622
QuantLib::Instruments::Swaption, 625
QuantLib::Instruments::Swaption
performCalculations, 625
setupArguments, 625
QuantLib::Instruments::Swaption::arguments, 627
QuantLib::Instruments::Swaption::results, 628
QuantLib::Instruments::VanillaOption, 664
QuantLib::Instruments::VanillaOption
impliedVolatility, 665
performCalculations, 666
setupArguments, 665
setupExpired, 665
QuantLib::Instruments::VanillaOption::arguments, 667
QuantLib::Instruments::VanillaOption::results, 668
QuantLib::IntegerFormatter, 425
QuantLib::Lattices::AdditiveEQPBinomialTree, 157
QuantLib::Lattices::BinomialTree, 203
QuantLib::Lattices::BlackScholesLattice, 213
QuantLib::Lattices::CoxRossRubinstein, 281
QuantLib::Lattices::EqualJumpsBinomialTree, 323
QuantLib::Lattices::EqualProbabilitiesBinomialTree, 324
QuantLib::Lattices::JarrowRudd, 433
QuantLib::Lattices::Lattice, 439
QuantLib::Lattices::Lattice
rollAlmostBack, 440
rollback, 440
QuantLib::Lattices::Lattice2D, 441
QuantLib::Lattices::Tian, 640
QuantLib::Lattices::Tree, 648
QuantLib::Lattices::Trigeorgis, 654
QuantLib::Lattices::TrinomialBranching, 655
QuantLib::Lattices::TrinomialTree, 656
QuantLib::Link, 455
QuantLib::Link
Link, 455
linkTo, 455
QuantLib::LocalVolTermStructure, 463
QuantLib::MakeSchedule, 469
QuantLib::MarketElement, 470
QuantLib::Math, 131
QuantLib::Math
hypot, 134
QuantLib::Math::BicubicSplineInterpolation, 191
QuantLib::Math::BicubicSplineInterpolation
operator(), 191
QuantLib::Math::BilinearInterpolation, 193
QuantLib::Math::BilinearInterpolation
operator(), 193
QuantLib::Math::CubicSpline, 283
QuantLib::Math::CubicSplineInterpolation, 284
QuantLib::Math::CubicSplineInterpolation
operator(), 284
QuantLib::Math::CumulativeNormalDistribution, 285
QuantLib::Math::DiscrepancyStatistics, 305
QuantLib::Math::ErrorFunction, 326
QuantLib::Math::GammaFunction, 374
QuantLib::Math::GaussianStatistics, 375
QuantLib::Math::GaussianStatistics
gaussianDownsideDeviation, 375
gaussianDownsideVariance, 375
gaussianExpectedShortfall, 376
gaussianPercentile, 376
gaussianPotentialUpside, 376
gaussianRegret, 375
gaussianValueAtRisk, 376
QuantLib::Math::GeneralStatistics, 378
QuantLib::Math::GeneralStatistics
add, 380
errorEstimate, 379
expectationValue, 380
kurtosis, 379
max, 380
mean, 379
min, 379
percentile, 380
skewness, 379
standardDeviation, 379
topPercentile, 380
variance, 379
QuantLib::Math::GenericRiskStatistics, 383
QuantLib::Math::GenericRiskStatistics
averageShortfall, 385
downsideDeviation, 384
downsideVariance, 383

expectedShortfall, 384
potentialUpside, 384
regret, 384
semiDeviation, 383
semiVariance, 383
shortfall, 384
valueAtRisk, 384
QuantLib::Math::IncrementalStatistics, 414
QuantLib::Math::IncrementalStatistics
add, 416
downsideDeviation, 415
downsideVariance, 415
errorEstimate, 415
kurtosis, 416
max, 416
mean, 415
min, 416
skewness, 416
standardDeviation, 415
variance, 415
QuantLib::Math::Interpolation, 427
QuantLib::Math::Interpolation
operator(), 428
QuantLib::Math::Interpolation2D, 429
QuantLib::Math::Interpolation2D
operator(), 430
QuantLib::Math::InverseCumulativeNormal, 431
QuantLib::Math::KronrodIntegral, 438
QuantLib::Math::LexicographicalView, 449
QuantLib::Math::Linear, 451
QuantLib::Math::LinearInterpolation, 452
QuantLib::Math::LinearInterpolation
operator(), 452
QuantLib::Math::LogLinear, 465
QuantLib::Math::LogLinearInterpolation, 466
QuantLib::Math::LogLinearInterpolation
operator(), 466
QuantLib::Math::Matrix, 471
QuantLib::Math::Matrix
matrixSqrt, 473
operator+=, 473
pseudoSqrt, 473
QuantLib::Math::MoroInverseCumulativeNormal, 505
QuantLib::Math::MultivariateAccumulator, 509
QuantLib::Math::NormalDistribution, 518
QuantLib::Math::PrimeNumbers, 560
QuantLib::Math::RiskMeasures, 583
QuantLib::Math::RiskMeasures
gaussianExpectedShortfall, 584
gaussianPercentile, 583
gaussianPotentialUpside, 583
gaussianValueAtRisk, 584
percentile, 583
potentialUpside, 583
valueAtRisk, 583
QuantLib::Math::SegmentIntegral, 590
QuantLib::Math::SequenceStatistics, 591
QuantLib::Math::SimpsonIntegral, 604
QuantLib::Math::SVD, 620
QuantLib::Math::SymmetricSchurDecomposition, 632
QuantLib::Math::TrapezoidIntegral, 646
QuantLib::MonteCarlo, 135
QuantLib::MonteCarlo
getCovariance, 137
QuantLib::MonteCarlo::ArithmeticAPOPathPricer_-old, 170
QuantLib::MonteCarlo::ArithmeticASOPathPricer_-old, 171
QuantLib::MonteCarlo::BarrierPathPricer, 187
QuantLib::MonteCarlo::BasketPathPricer_-old, 188
QuantLib::MonteCarlo::BiasedBarrierPathPricer, 190
QuantLib::MonteCarlo::BinaryPathPricer, 202
QuantLib::MonteCarlo::BrownianBridge, 231
QuantLib::MonteCarlo::CliquetOptionPathPricer, 261
QuantLib::MonteCarlo::CliquetOptionPathPricer_-old, 262
QuantLib::MonteCarlo::EuropeanPathPricer, 331
QuantLib::MonteCarlo::EuropeanPathPricer_-old, 332
QuantLib::MonteCarlo::EverestPathPricer_-old, 333
QuantLib::MonteCarlo::GeometricAPOPathPricer_-old, 386
QuantLib::MonteCarlo::GeometricASOPathPricer_-old, 387
QuantLib::MonteCarlo::HimalayaPathPricer_-old, 393
QuantLib::MonteCarlo::MaxBasketPathPricer_-old, 474
QuantLib::MonteCarlo::MonteCarloModel, 504
QuantLib::MonteCarlo::MultiPath, 506
QuantLib::MonteCarlo::MultiPathGenerator, 507
QuantLib::MonteCarlo::MultiPathGenerator_-old, 508

QuantLib::MonteCarlo::PagodaPathPricer_-
old, 538
QuantLib::MonteCarlo::Path, 543
QuantLib::MonteCarlo::PathGenerator, 544
QuantLib::MonteCarlo::PathGenerator_old,
545
QuantLib::MonteCarlo::PathGenerator_old
PathGenerator_old, 545
QuantLib::MonteCarlo::PathPricer, 546
QuantLib::MonteCarlo::PathPricer_old, 547
QuantLib::MonteCarlo::PerformanceOptionPathPricer_-
old, 550
QuantLib::MonteCarlo::Sample, 587
QuantLib::Null, 519
QuantLib::NumericalMethod, 522
QuantLib::Optimization::ArmijoLineSearch,
172
QuantLib::Optimization::BoundaryConstraint,
226
QuantLib::Optimization::CompositeConstraint,
266
QuantLib::Optimization::ConjugateGradient,
268
QuantLib::Optimization::Constraint, 270
QuantLib::Optimization::ConstraintImpl,
271
QuantLib::Optimization::CostFunction, 273
QuantLib::Optimization::EndCriteria, 321
QuantLib::Optimization::LeastSquareFunction,
446
QuantLib::Optimization::LeastSquareProblem,
447
QuantLib::Optimization::LeastSquare-
Problem
targetValueAndGradient, 447
QuantLib::Optimization::LineSearch, 453
QuantLib::Optimization::Method, 497
QuantLib::Optimization::NoConstraint, 515
QuantLib::Optimization::NonLinearLeastSquare,
516
QuantLib::Optimization::PositiveConstraint,
556
QuantLib::Optimization::Problem, 561
QuantLib::Optimization::Simplex, 603
QuantLib::Optimization::Simplex
Simplex, 603
QuantLib::Optimization::SteepestDescent,
611
QuantLib::Option, 533
QuantLib::OrnsteinUhlenbeckProcess, 534
QuantLib::OrnsteinUhlenbeckProcess
expectation, 534
variance, 534
QuantLib::OutOfMemoryError, 537
QuantLib::Patterns, 138
QuantLib::Patterns::AcyclicVisitor, 156
QuantLib::Patterns::Bridge, 230
QuantLib::Patterns::CuriouslyRecurringTemplate,
286
QuantLib::Patterns::LazyObject, 443
QuantLib::Patterns::LazyObject
calculate, 444
freeze, 444
performCalculations, 444
recalculate, 444
unfreeze, 444
update, 444
QuantLib::Patterns::Observable, 523
QuantLib::Patterns::Observable
notifyObservers, 524
QuantLib::Patterns::Observer, 525
QuantLib::Patterns::Observer
update, 527
QuantLib::Patterns::Visitor, 672
QuantLib::Payoff, 548
QuantLib::Period, 551
QuantLib::PlainVanillaPayoff, 555
QuantLib::PostconditionNotSatisfiedError,
557
QuantLib::PreconditionNotSatisfiedError,
558
QuantLib::Pricers, 139
QuantLib::Pricers::AnalyticalCapFloor, 164
QuantLib::Pricers::BarrierOption, 184
QuantLib::Pricers::BinaryOption, 199
QuantLib::Pricers::BlackCapFloor, 206
QuantLib::Pricers::BlackSwaption, 215
QuantLib::Pricers::CliquetOption, 259
QuantLib::Pricers::ContinuousGeometricAPO,
272
QuantLib::Pricers::DiscreteGeometricAPO,
306
QuantLib::Pricers::DiscreteGeometricASO,
307
QuantLib::Pricers::EuropeanOption, 330
QuantLib::Pricers::FdAmericanOption, 343
QuantLib::Pricers::FdBermudanOption, 344
QuantLib::Pricers::FdBsmOption, 345
QuantLib::Pricers::FdDividendEuropeanOption,
347
QuantLib::Pricers::FdDividendShoutOption,
348
QuantLib::Pricers::FdEuropean, 349
QuantLib::Pricers::FdStepConditionOption,
350
QuantLib::Pricers::JamshidianSwaption, 432
QuantLib::Pricers::McBasket, 477
QuantLib::Pricers::McCliquetOption, 480

QuantLib::Pricers::McDiscreteArithmeticAPO, 481
QuantLib::Pricers::McDiscreteArithmeticASO, 482
QuantLib::Pricers::McEuropean, 483
QuantLib::Pricers::McEverest, 485
QuantLib::Pricers::McHimalaya, 486
QuantLib::Pricers::McMaxBasket, 487
QuantLib::Pricers::McPagoda, 488
QuantLib::Pricers::McPerformanceOption, 489
QuantLib::Pricers::McPricer, 490
QuantLib::Pricers::PerformanceOption, 549
QuantLib::Pricers::SingleAssetOption, 605
QuantLib::Pricers::SingleAssetOption
impliedVolatility, 606
QuantLib::Pricers::TreeCapFloor, 649
QuantLib::Pricers::TreeSwaption, 650
QuantLib::PricingEngine, 559
QuantLib::PricingEngines::AmericanMCVanillaEngine, 163
QuantLib::PricingEngines::AnalyticAmericanBinaryEngine, 165
QuantLib::PricingEngines::AnalyticBarrierEngine, 166
QuantLib::PricingEngines::AnalyticEuropeanBinaryEngine, 167
QuantLib::PricingEngines::AnalyticEuropeanEngine, 168
QuantLib::PricingEngines::BarrierEngine, 181
QuantLib::PricingEngines::BinaryEngine, 196
QuantLib::PricingEngines::BinomialVanillaEngine, 204
QuantLib::PricingEngines::CliquetEngine, 257
QuantLib::PricingEngines::ForwardEngine, 358
QuantLib::PricingEngines::ForwardPerformanceEngine, 360
QuantLib::PricingEngines::GenericEngine, 381
QuantLib::PricingEngines::GenericModelEngine, 382
QuantLib::PricingEngines::GenericModel-Engine
update, 382
QuantLib::PricingEngines::IntegralEngine, 426
QuantLib::PricingEngines::LatticeShortRateModelEngine, 442
QuantLib::PricingEngines::LatticeShort-RateModelEngine
update, 442
QuantLib::PricingEngines::MCBarrierEngine, 475
QuantLib::PricingEngines::MCBinaryEngine, 478
QuantLib::PricingEngines::MCEuropeanEngine, 484
QuantLib::PricingEngines::McSimulation, 492
QuantLib::PricingEngines::MCVanillaEngine, 494
QuantLib::PricingEngines::QuantoEngine, 565
QuantLib::PricingEngines::VanillaEngine, 663
QuantLib::RandomNumbers, 142
QuantLib::RandomNumbers::BoxMullerGaussianRng, 227
QuantLib::RandomNumbers::BoxMuller-GaussianRng
BoxMullerGaussianRng, 227
QuantLib::RandomNumbers::CLGaussianRng, 256
QuantLib::RandomNumbers::CLGaussian-Rng
CLGaussianRng, 256
QuantLib::RandomNumbers::HaltonRsg, 389
QuantLib::RandomNumbers::ICGaussianRng, 404
QuantLib::RandomNumbers::ICGaussian-Rng
ICGaussianRng, 404
QuantLib::RandomNumbers::ICGaussianRsg, 405
QuantLib::RandomNumbers::KnuthUniformRng, 437
QuantLib::RandomNumbers::Knuth-UniformRng
KnuthUniformRng, 437
next, 437
QuantLib::RandomNumbers::LecuyerUniformRng, 448
QuantLib::RandomNumbers::Lecuyer-UniformRng
LecuyerUniformRng, 448
next, 448
QuantLib::RandomNumbers::MersenneTwisterUniformRng, 496
QuantLib::RandomNumbers::Mersenne-TwisterUniformRng
MersenneTwisterUniformRng, 496
next, 496

QuantLib::RandomNumbers::RandomArrayGenerator, 574
QuantLib::RandomNumbers::RandomSequenceGenerator, 575
QuantLib::RandomNumbers::SobolRsg, 607
QuantLib::RateFormatter, 576
QuantLib::RelinkableHandle, 579
QuantLib::RelinkableHandle
linkTo, 579
RelinkableHandle, 579
QuantLib::Results, 581
QuantLib::RiskStatistics_old, 585
QuantLib::RiskStatistics_old
add, 586
QuantLib::Schedule, 588
QuantLib::ShortRateModels, 144
QuantLib::ShortRateModels::AffineModel, 158
QuantLib::ShortRateModels::BlackKarasinski, 209
QuantLib::ShortRateModels::BlackKarasinski::Dynamics, 210
QuantLib::ShortRateModels::CalibrationHelper, 240
QuantLib::ShortRateModels::CalibrationHelper
update, 241
QuantLib::ShortRateModels::CalibrationSet, 242
QuantLib::ShortRateModels::ConstantParameter, 269
QuantLib::ShortRateModels::CoxIngersollRoss, 278
QuantLib::ShortRateModels::CoxIngersollRoss::Dynamics, 280
QuantLib::ShortRateModels::ExtendedCoxIngersollRoss, 336
QuantLib::ShortRateModels::ExtendedCoxIngersollRoss::Dynamics, 338
QuantLib::ShortRateModels::ExtendedCoxIngersollRoss::FittingParameter, 339
QuantLib::ShortRateModels::G2, 371
QuantLib::ShortRateModels::G2::FittingParameter, 373
QuantLib::ShortRateModels::HullWhite, 400
QuantLib::ShortRateModels::HullWhite::Dynamics, 402
QuantLib::ShortRateModels::HullWhite::FittingParameter, 403
QuantLib::ShortRateModels::Model, 502
QuantLib::ShortRateModels::Model
calibrate, 503
update, 503
QuantLib::ShortRateModels::NullParameter, 521
QuantLib::ShortRateModels::OneFactorAffineModel, 528
QuantLib::ShortRateModels::OneFactorModel, 529
QuantLib::ShortRateModels::OneFactorModel::ShortRateDynamics, 530
QuantLib::ShortRateModels::OneFactorModel::ShortRateTree, 531
QuantLib::ShortRateModels::Parameter, 539
QuantLib::ShortRateModels::ParameterImpl, 540
QuantLib::ShortRateModels::PiecewiseConstantParameter, 552
QuantLib::ShortRateModels::TermStructureConsistentModel, 637
QuantLib::ShortRateModels::TermStructureFittingParameter, 638
QuantLib::ShortRateModels::TwoFactorModel, 657
QuantLib::ShortRateModels::TwoFactorModel::ShortRateDynamics, 658
QuantLib::ShortRateModels::TwoFactorModel::ShortRateTree, 659
QuantLib::ShortRateModels::Vasicek, 669
QuantLib::ShortRateModels::Vasicek::Dynamics, 671
QuantLib::SimpleMarketElement, 598
QuantLib::Solver1D, 608
QuantLib::Solver1D
setMaxEvaluations, 609
solve, 609
QuantLib::Solvers1D, 147
QuantLib::Solvers1D::Bisection, 205
QuantLib::Solvers1D::Brent, 229
QuantLib::Solvers1D::FalsePosition, 342
QuantLib::Solvers1D::Newton, 512
QuantLib::Solvers1D::NewtonSafe, 513
QuantLib::Solvers1D::Ridder, 582
QuantLib::Solvers1D::Secant, 589
QuantLib::SquareRootProcess, 610
QuantLib::StrikedTypePayoff, 617
QuantLib::StringFormatter, 618
QuantLib::SupersharePayoff, 619
QuantLib::SwaptionVolatilityStructure, 630
QuantLib::TermStructure, 634
QuantLib::TermStructures, 148
QuantLib::TermStructures::AffineTermStructure, 159
QuantLib::TermStructures::AffineTermStructure
update, 160

QuantLib::TermStructures::DepositRateHelper, 295
QuantLib::TermStructures::DepositRateHelper
setTermStructure, 295
QuantLib::TermStructures::DiscountCurve, 301
QuantLib::TermStructures::DriftTermStructure, 318
QuantLib::TermStructures::DriftTermStructure
update, 319
QuantLib::TermStructures::ExtendedDiscountCurve, 340
QuantLib::TermStructures::ExtendedDiscountCurve
compoundForwardImpl, 341
update, 341
QuantLib::TermStructures::ForwardSpreadedTermStructure, 363
QuantLib::TermStructures::ForwardSpreadedTermStructure
update, 364
zeroYieldImpl, 364
QuantLib::TermStructures::FraRateHelper, 368
QuantLib::TermStructures::FraRateHelper
setTermStructure, 368
QuantLib::TermStructures::FuturesRateHelper, 370
QuantLib::TermStructures::ImpliedTermStructure, 409
QuantLib::TermStructures::ImpliedTermStructure
update, 410
QuantLib::TermStructures::PiecewiseFlatForward, 553
QuantLib::TermStructures::PiecewiseFlatForward
PiecewiseFlatForward, 554
QuantLib::TermStructures::QuantoTermStructure, 570
QuantLib::TermStructures::QuantoTermStructure
update, 571
QuantLib::TermStructures::RateHelper, 577
QuantLib::TermStructures::RateHelper
setTermStructure, 578
update, 578
QuantLib::TermStructures::SwapRateHelper, 623
QuantLib::TermStructures::SwapRateHelper
setTermStructure, 624
QuantLib::TermStructures::ZeroCurve, 679
QuantLib::TermStructures::ZeroSpreadedTermStructure, 681
QuantLib::TermStructures::ZeroSpreadedTermStructure
forwardImpl, 682
update, 682
QuantLib::TimeGrid, 642
QuantLib::TimeGrid
TimeGrid, 642
QuantLib::Utilities, 150
QuantLib::Utilities::combining_iterator, 264
QuantLib::Utilities::coupling_iterator, 274
QuantLib::Utilities::filtering_iterator, 351
QuantLib::Utilities::lowest_category_iterator, 468
QuantLib::Utilities::processing_iterator, 563
QuantLib::Utilities::stepping_iterator, 613
QuantLib::Value, 662
QuantLib::Volatilities::CapFlatVolatilityVector, 246
QuantLib::Volatilities::SwaptionVolatilityMatrix, 629
QuantLib::VolTermStructures, 151
QuantLib::VolTermStructures::BlackConstantVol, 207
QuantLib::VolTermStructures::BlackConstantVol
update, 208
QuantLib::VolTermStructures::BlackVarianceCurve, 216
QuantLib::VolTermStructures::BlackVarianceCurve
update, 217
QuantLib::VolTermStructures::BlackVarianceSurface, 218
QuantLib::VolTermStructures::BlackVarianceSurface
update, 219
QuantLib::VolTermStructures::ImpliedVolTermStructure, 411
QuantLib::VolTermStructures::ImpliedVolTermStructure
update, 412
QuantLib::VolTermStructures::LocalConstantVol, 457
QuantLib::VolTermStructures::LocalConstantVol
update, 458
QuantLib::VolTermStructures::LocalVolCurve, 459
QuantLib::VolTermStructures::LocalVolCurve
localVolImpl, 460

update, 460
QuantLib::VolTermStructures::LocalVolSurface, 461
QuantLib::VolTermStructures::LocalVolSurface
update, 462
QuantLib::ZeroYieldStructure, 683
QuantLib::ZeroYieldStructure
compoundForwardImpl, 683
discountImpl, 683
forwardImpl, 683

recalculate
QuantLib::Patterns::LazyObject, 444
regret
QuantLib::Math::GenericRiskStatistics, 384
RelinkableHandle
QuantLib::RelinkableHandle, 579
ROL
QuantLib, 116
roll
QuantLib::Calendar, 237
rollAlmostBack
QuantLib::Lattices::Lattice, 440
rollback
QuantLib::FiniteDifferences::FiniteDifferenceModel, 352
QuantLib::Lattices::Lattice, 440
RollingConvention
QuantLib, 115

SAR
QuantLib, 116
Scheduler
QuantLib, 114
SEK
QuantLib, 116
semiDeviation
QuantLib::Math::GenericRiskStatistics, 383
semiVariance
QuantLib::Math::GenericRiskStatistics, 383
sensitivity
QuantLib::Instruments::Swap, 622
sequencestatistics.hpp
DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE, 888
DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID, 888
setMaxEvaluations
QuantLib::Solver1D, 609
setPricingEngine
QuantLib::Instrument, 423
setTermStructure
QuantLib::TermStructures::DepositRateHelper, 295
QuantLib::TermStructures::FraRateHelper, 368
QuantLib::TermStructures::RateHelper, 578
QuantLib::TermStructures::SwapRateHelper, 624
setTime
QuantLib::FiniteDifferences::BoundaryCondition, 225
QuantLib::FiniteDifferences::DirichletBC, 300
QuantLib::FiniteDifferences::NeumannBC, 511
setupArguments
QuantLib::Instrument, 423
QuantLib::Instruments::BarrierOption, 183
QuantLib::Instruments::BinaryOption, 198
QuantLib::Instruments::CapFloor, 248
QuantLib::Instruments::ForwardVanillaOption, 366
QuantLib::Instruments::QuantoForwardVanillaOption, 567
QuantLib::Instruments::QuantoVanillaOption, 573
QuantLib::Instruments::SimpleSwap, 600
QuantLib::Instruments::Swaption, 625
QuantLib::Instruments::VanillaOption, 665
setupExpired
QuantLib::Instrument, 423
QuantLib::Instruments::BarrierOption, 183
QuantLib::Instruments::BinaryOption, 198
QuantLib::Instruments::QuantoVanillaOption, 573
QuantLib::Instruments::Swap, 622
QuantLib::Instruments::VanillaOption, 665
SGD
QuantLib, 116
shortfall
QuantLib::Math::GenericRiskStatistics, 384
Side
QuantLib::FiniteDifferences::BoundaryCondition, 224

SimpleSwap
QuantLib::Instruments::SimpleSwap, 600
Simplex
QuantLib::Optimization::Simplex, 603
SIT
QuantLib, 116
skewness
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 416
SKK
QuantLib, 116
solve
QuantLib::Solver1D, 609
standardDeviation
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 415
String functions, 100

targetValueAndGradient
QuantLib::Optimization::LeastSquareProblem, 447
Template capabilities, 104
templateMacros
QL_ALLOW_TEMPLATE_METHOD_CALLS, 104
QL_DECLARE_TEMPLATE_SPECIALIZATIONS, 104
QL_TEMPLATE_METAPROGRAMMING_WORKS, 104
QL_TYPENAME, 104
THB
QuantLib, 116
Time functions, 99
TimeGrid
QuantLib::TimeGrid, 642
topPercentile
QuantLib::Math::GeneralStatistics, 380
TRL
QuantLib, 116
TTD
QuantLib, 116
TWD
QuantLib, 116

unfreeze
QuantLib::Patterns::LazyObject, 444
update
QuantLib::BlackModel, 211
QuantLib::CashFlows::IndexedCoupon, 420
QuantLib::CashFlows::ParCoupon, 542
QuantLib::CompositeMarketElement, 267
QuantLib::DerivedMarketElement, 297
QuantLib::Indexes::Xibor, 676
QuantLib::Patterns::LazyObject, 444
QuantLib::Patterns::Observer, 527
QuantLib::PricingEngines::GenericModelEngine, 382
QuantLib::PricingEngines::LatticeShortRateModelEngine, 442
QuantLib::ShortRateModels::CalibrationHelper, 241
QuantLib::ShortRateModels::Model, 503
QuantLib::TermStructures::AffineTermStructure, 160
QuantLib::TermStructures::DriftTermStructure, 319
QuantLib::TermStructures::ExtendedDiscountCurve, 341
QuantLib::TermStructures::ForwardSpreadedTermStructure, 364
QuantLib::TermStructures::ImpliedTermStructure, 410
QuantLib::TermStructures::QuantoTermStructure, 571
QuantLib::TermStructures::RateHelper, 578
QuantLib::TermStructures::ZeroSpreadedTermStructure, 682
QuantLib::VolTermStructures::BlackConstantVol, 208
QuantLib::VolTermStructures::BlackVarianceCurve, 217
QuantLib::VolTermStructures::BlackVarianceSurface, 219
QuantLib::VolTermStructures::ImpliedVolTermStructure, 412
QuantLib::VolTermStructures::LocalConstantVol, 458
QuantLib::VolTermStructures::LocalVolCurve, 460
QuantLib::VolTermStructures::LocalVolSurface, 462
USD
QuantLib, 116

valueAtRisk
QuantLib::Math::GenericRiskStatistics, 384
QuantLib::Math::RiskMeasures, 583
VanillaCap
QuantLib::Instruments, 130

VanillaCapFloor
QuantLib::Instruments, 130
VanillaCollar
QuantLib::Instruments, 130
VanillaFloor
QuantLib::Instruments, 130
variance
QuantLib::DiffusionProcess, 298
QuantLib::Math::GeneralStatistics, 379
QuantLib::Math::IncrementalStatistics, 415
QuantLib::OrnsteinUhlenbeckProcess, 534
VEB
QuantLib, 116

Weekday
QuantLib, 116

ZAR
QuantLib, 116
zeroYieldImpl
QuantLib::DiscountStructure, 303
QuantLib::ForwardRateStructure, 361
QuantLib::TermStructures::ForwardSpreadedTermStructure, 364